Sistema di navigazione per robot mobili basato sull'anticipazione sensoriale

Alessandro Manzi Matteo Gabelletti

Università degli Studi di Pisa

FACOLTÀ DI SCIENZE MATEMATICHE FISICHE NATURALI

DIPARTIMENTO DI INFORMATICA

Corso di Laurea Specialistica in Tecnologie Informatiche

Sistema di navigazione per robot mobili basato sull'anticipazione sensoriale

Candidati:

Alessandro Manzi

Matteo Gabelletti

Relatore:

Prof.ssa Cecilia Laschi

Anno Accademico 2006–2007

Indice

Elenco delle figure

Capitolo 1

Introduzione

Da alcuni decenni la ricerca nel campo della robotica sta vivendo un incremento di interesse sia in ambito accademico che industriale e le tecnologie robotiche hanno avuto una rapida evoluzione. Si è passati dalle applicazioni industriali di robot statici in ambienti strutturati a robot mobili immersi in ambienti dinamici e parzialmente conosciuti. Particolare interesse ha rivestito anche l'interazione con l'uomo. Per esempio, sono nati ambiti di ricerca nel campo della riabilitazione, nell'assistenza agli anziani e disabili, nelle applicazioni domestiche e nell'industria dell'intrattenimento. Nel campo dell'informatica gli argomenti di interesse riguardano l'intelligenza artificiale ed in particolare lo studio dell'interazione con l'ambiente acquisendo dati sensoriali, elaborandoli e generando comandi per l'interazione del robot nel mondo reale. Sotto questo punto di vista l'informatica, nel corso degli anni, ha sviluppato tutta una serie di algoritmi atti a generare una risposta in base alle percezioni ricevute dall'esterno.

Una delle ultime evoluzioni nell'ambito della ricerca in robotica è lo sviluppo di modelli bioispirati. I modelli robotici bioispirati tentano di replicare soluzioni trovate dalla natura per risolvere problemi comuni anche all'ambito

robotico. Una branca della robotica bioispirata è la neurorobotica, la quale si occupa dello studio dei meccanismi di elaborazione sensoriale e di sintesi del comportamento (coordinazione senso-motoria) nell'uomo e nei sistemi robotici, ed è fondamentale sia per il progresso della robotica che per la comprensione del funzionamento del cervello umano (neuroscienze).

Proprio l'ispirazione ai meccanismi umani di controllo motorio e coordinazione senso-motoria e l'implementazione di modelli del cervello hanno portato recentemente alla formulazione di paradigmi anticipatori per il controllo del comportamento di robot, basati su predizioni degli input percettivi. Infatti, dallo studio delle reazioni senso-motorie nell'uomo si è scoperto che tali reazioni sono troppo rapide da poter essere spiegate con un meccanismo a feedback sensoriale, e sono tali invece da essere giustificate da una qualche predizione sensoriale.

Il lavoro di tesi riguarda proprio lo studio e l'implementazione di un paradigma anticipatorio, in un problema di navigazione autonoma per un robot mobile in ambiente esterno, dinamico e parzialmente conosciuto. Poter fare delle predizioni sulla risposta sensoriale in un ambiente parzialmente conosciuto significa avere a priori o costruire una rappresentazione interna del mondo, che però, essendo questo dinamico può variare nel tempo.

Tipicamente un sistema di navigazione è composto da un modulo per la lettura dei dati sensoriali, un modulo per l'elaborazione degli stessi, un modulo per la pianificazione delle azioni e un modulo di generazione dei comandi agli attuatori del robot.

Il modulo dell'elaborazione dei dati sensoriali ha il compito di tradurre i dati dei sensori in informazioni fruibili dal modulo di pianificazione.

Il modulo di pianificazione definisce le azioni da compiere per raggiungere l'obiettivo prefissato. Per fare questo, esso genera una serie di sotto-obiettivi

che eventualmente vengono aggiornati in presenza di problemi imprevisti.

Il sistema di navigazione dovrà implementare un algoritmo per la costruzione della mappa, uno per la localizzazione e uno per la creazione del path ottimale verso l'obiettivo. L'algoritmo di costruzione della mappa realizza un modello interno del mondo nel quale il robot si troverà ad operare. L'algoritmo scelto per questa tesi è l'Histogrammic in Motion Mapping (HIMM), che è il più utilizzato per risolvere questo tipo di problemi per le sue caratteristiche di velocità e leggerezza.

L'algoritmo di localizzazione serve per sapere dove si trova il robot all'interno della mappa. L'algoritmo di creazione del path realizza una lista ottimale di sotto-obiettivi da raggiungere per completare il proprio compito. Per fare questo abbiamo utilizzato l'algoritmo A*, anche conosciuto come algoritmo di Dijkstra, generalmente utilizzato per la generazione di percorsi minimi su grafi.

L'aspetto innovativo di questa tesi è la realizzazione di un modulo di anticipazione che, in base a dei modelli interni, prevede la risposta sensoriale del robot nella posizione in cui si troverà prossimamente, in base al comando motorio corrente.

Questo approccio permette di anticipare i calcoli per l'elaborazione dei dati sensoriali e successivamente confrontarli con i dati sensoriali non appena questi sono acquisiti dall'ambiente esterno. Nel caso in cui il confronto sia positivo, l'elaborazione dei dati stessi è evitata e si passa direttamente alla generazione dei comandi successivi per raggiungere i sotto-obiettivi pianificati precedentemente. Nel caso in cui il confronto faccia emergere delle incoerenze tra la percezione attesa e quella reale, e quindi ci si trovi in presenza di eventi imprevisti, il sistema procede alla elaborazione dei dati sensoriali e alla pianificazione delle prossime azioni motorie, come nel caso dei comportamenti

reattivi.

Siccome l'elaborazione dei dati sensoriali in genere è laboriosa e complessa, evitarla, comporta un miglioramento complessivo delle prestazioni del sistema; infatti ci si aspetta che il robot possa procedere più velocemente pur essendo reattivo in caso di ostacoli imprevisti, visto che il sistema è più rapido nell'eseguire l'elaborazione sensoriale.

Nel capitolo 2 verrà fatta una panoramica sulle architetture classiche della robotica; si parlerà del paradigma gerarchico, reattivo , ibrido. In più verrà descritto in maggior dettaglio il paradigma anticipatorio, che permette la predizione dei dati sensoriali. Partendo da nozioni teoriche verrano esposti alcuni esempi concreti.

Nel capitolo 3 verrà spiegata l'architettura classica del sistema di navigazione. Saranno esposte le strutture dati per la realizzazione di un modello interno e gli algoritmi per generarli. Verranno inoltre descritti algoritmi per la realizzazione di path ottimali, per la localizzazione e paradigmi di 'obstacle avoidance'.

Nel capitolo 4 vengono forniti i dettagli implementativi di una applicazione di navigazione della mappa e di una per la navigazione. Per quest'ultima applicazione viene descritta una versione che fa uso di un sistema di navigazione classico e una che utilizza il paradigma anticipatorio.

Nel Capitolo 5, verranno descritti i dettagli hardware e software del robot B21r utilizzato per eseguire le prove sugli applicativi descritti nel capitolo 4. Più specificatamente si entrerà nel dettaglio del funzionamento dei sensori a ultrasuoni e nell'architettura del Mobility, il software di gestione fornito dalla iRobot la casa di costruzione del B21r.

Nel Capitolo 6, sono descritte le prove sperimentali eseguite sulle varie applicazioni realizzate, utilizzando il robot in un ambiente reale e sono discussi

i risultati ottenuti. Queste prove sono state realizzate in ambiente esterno, opportunamente dotato di ostacoli per il robot. Il tutto è corredato da una sequenza fotografica che mostra una delle prove effettuate.

Nel Capitolo 7, partendo dall' analisi dei risultati, vengono tratte alcune conclusioni sui risultati ottenuti e fatte delle ipotesi sugli sviluppi futuri nella navigazione autonoma basata su anticipazione sensoriale.

Capitolo 2

Architetture dei sistemi robotici

Nel corso degli anni la robotica ha subìto molti cambiamenti trovando numerose applicazioni in vari ambiti della vita dell'uomo, evolvendosi e adattandosi mano a mano alle nuove esigenze. Troviamo, ad esempio, applicazioni nell'industria [SS00], nella medicina [Neu], in ambiente domestico, militare e spaziale. [Bek05]

Le architetture robotiche sono quindi varie e dipendenti dal campo di applicazione e dagli scopi che si vogliono ottenere. E' possibile comunque definire uno schema abbastanza generale come quello mostrato in figura 2.1[Mur00], in cui risulta chiaro che un robot non è qualcosa di isolato, ma è immerso in un mondo, col quale interagisce attraverso i suoi attuatori e dal quale riceve informazioni tramite i propri sensori. Tra sensori ed attuatori c'è un modulo molto importante che si occupa della pianificazione, la quale sostanzialmente modella il comportamento generale del robot.

Da questo schema possiamo distinguere due livelli per quanto riguarda il controllo di un robot:

- Controllori di basso livello, che agiscono direttamente sull'effettore per

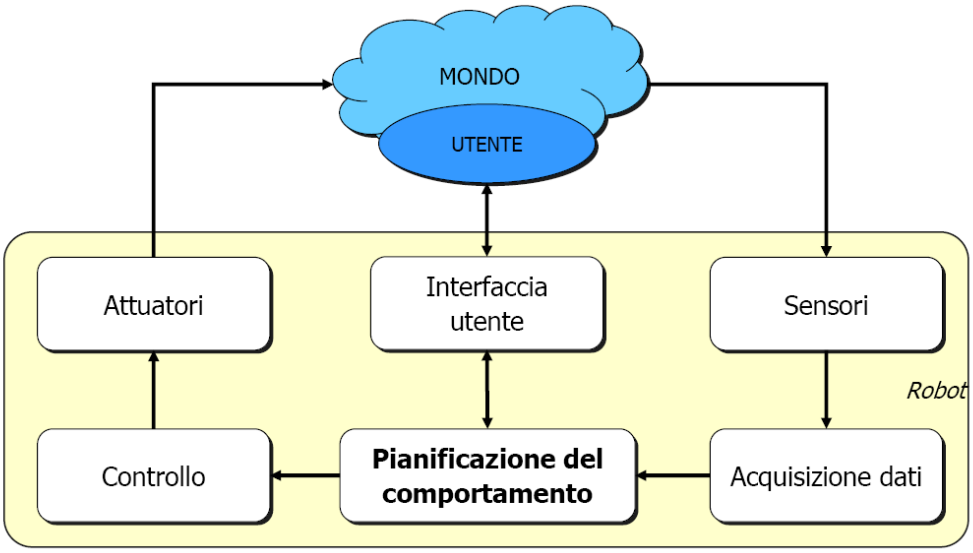

Figura 2.1: Schema tipico di un sistema robotico

controllarne la dinamica (gli attuatori e il modulo del controllo della figura 2.1);

- Controllori di alto livello, (come ad esempio supervisori, pianificatori) che hanno il compito di pianificare e supervisionare il comportamento complessivo del robot (il modulo di pianificazione e l'interfaccia utente della figura 2.1).

Non esiste un'unica definizione di robotica e nel corso degli anni è cambiato molto anche il suo significato. Dato che uno dei primi campi di applicazione della robotica è stato appunto nell'automazione industriale, una delle primissime definizioni è stata data proprio dalla *Robotics Industry Association* e risale agli anni '80. Viene definita in questo modo:

Un robot è un manipolatore multifunzionale riprogrammabile progettato per muovere materiali, componenti, o dispositivi specializzati, attraverso movi-

menti variabili programmati per lo svolgimento del compito.

Le caratteristiche che erano (e sono tuttora) richieste ad un robot industriale sono l'accuratezza, la velocità e la ripetitività delle azioni in un ambiente di lavoro strutturato e noto a priori.

Leggendo la precedente definizione si capisce come essa poco si adatti a robot mobili che debbano agire fuori dall'ambiente industriale. Gli obiettivi e le caratteristiche richieste sono molto differenti, sono necessarie maggiori capacità percettive e soprattutto è fondamentale un comportamento reattivo idoneo agli ambienti dinamici, propri del "mondo reale", in cui agiscono abitualmente gli esseri umani.

Tra le definizioni che si adeguano ad un robot autonomo è la seguente:

Macchina capace di accettare ed eseguire autonomamente comandi o missioni in ambienti non completamente strutturati senza l'intervento dell'uomo. [Mur00]

Il problema principale è quindi quello di dover pianificare dinamicamente i comportamenti del robot in un ambiente di lavoro non noto a priori e variabile nel tempo, in funzione della richiesta di esecuzione di un determinato compito.

Quello che si rende necessario a questo punto, è creare un modulo di supervisione e pianificazione del comportamento. Nel corso degli anni sono stati sviluppati vari paradigmi per affrontare al meglio ogni tipologia di problema.

2.1 Paradigmi di pianificazione

Un paradigma è un insieme di assunzioni e tecniche che caratterizzano l'intero approccio ad una classe di problemi. In tal senso, nessun paradigma è corretto; piuttosto esistono problemi che sembrano più facilmente trattabili seguendo un particolare approccio. In altre parole, applicare il paradigma più adatto semplifica la soluzione del problema.

Ogni paradigma di pianificazione si basa sull'organizzazione delle principali funzioni primitive individuate nei seguenti moduli:

SENSE/Percezione prende come input i dati sensoriali e restituisce le informazioni percepite (processate)

PLAN/Pianificazione utilizzando alcune informazioni, sia percepite che cognitive (modello del mondo), produce delle direttive, delle decisioni

ACT/Azione,Attuazione da informazioni sensoriali o direttive seleziona i comandi da inviare agli attuatori

Queste tre primitive svolgono un ruolo essenziale nella definizione dei diversi approcci alla risoluzione di un problema di robotica. In particolare, possiamo descrivere i paradigmi di progettazione in ambito robotico mediante due metodologie ortogonali e complementari:

- Approccio basato sulle primitive: descrive le relazioni tra le tre primitive SENSE, PLAN e ACT;
- Approccio basato sul controllo e la distribuzione delle informazioni: descrive il modo in cui i dati acquisiti dai sensori vengono processati e distribuiti attraverso i vari componenti del sistema.

Oltre alla definizione di queste primitive bisogna caratterizzare i paradigmi in base all'organizzazione sensoriale in essi attuata. Per organizzazione sensoriale si intende proprio l'organizzazione delle componenti e l'attribuzione delle responsabilità di trattamento delle informazioni provenienti dai sensori.

I paradigmi storicamente più importanti [Ark98][Mur00], utilizzati in robotica per affrontare il problema della supervisione e della pianificazione del comportamento di un robot, sono:

- Paradigma gerarchico
- Paradigma reattivo
- Paradigma ibrido (deliberativo/reattivo)

Ogni paradigma influenza fortemente l' architettura software per l'implementazione della logica di controllo e dell' "intelligenza" di un robot.

2.1.1 Paradigma gerarchico

Nel paradigma gerarchico le funzioni primitive sono organizzate nella seguente maniera:

$$SENSE \longrightarrow PLAN \longrightarrow ACT$$

Il robot percepisce il mondo esterno, ossia l'ambiente su cui opera, pianifica la prossima azione da compiere ed, infine, esegue tale azione. Queste attività vengono svolte dal robot in modo strettamente sequenziale, si faccia riferimento a tal proposito alla figura 2.2. Al termine della fase di ACTING, il ciclo si ripete finché non viene raggiunto l'obiettivo prefissato. È facile

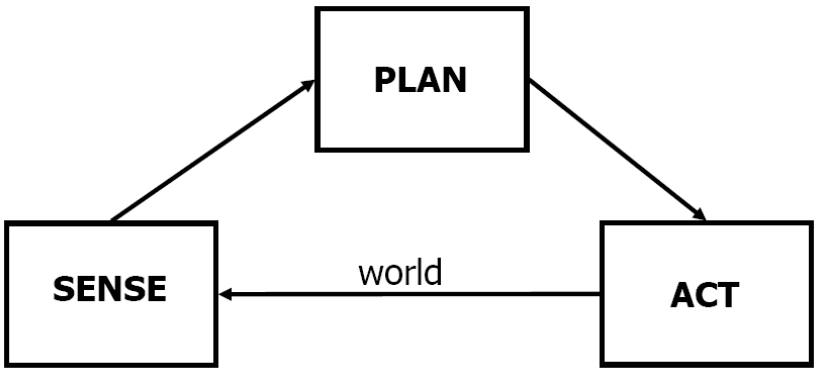

Figura 2.2: Paradigma gerarchico

intuire quindi che, in ogni architettura che implementa il paradigma gerarchico, esistono almeno tre componenti distinti: il modulo di acquisizione dei dati dai sensori, quello di pianificazione e quello di elaborazione dei comandi agli attuatori. Un'altra caratteristica che contraddistingue il paradigma gerarchico è relativa al trattamento dei dati percepiti dai sensori, i quali vengono elaborati al fine di formare un modello globale del mondo, ossia una singola rappresentazione che viene poi utilizzata dal modulo di pianificazione per decidere quale sarà la prossima azione da compiere.

Come rappresentato in figura 2.3[Bro86], il modulo cognitivo è utilizzato per interpretare la percezione e per pianificare le azioni degli attuatori.

Nelle architetture gerarchiche la percezione viene utilizzata per stabilire e mantenere una corrispondenza tra il modello interno del mondo (o mappa) e il mondo esterno.

Tipicamente il modello del mondo contiene:

- una rappresentazione a priori dell'ambiente in cui il robot opera
- l'informazione sensoriale percepita
- altre informazioni necessarie per l'esecuzione del compito

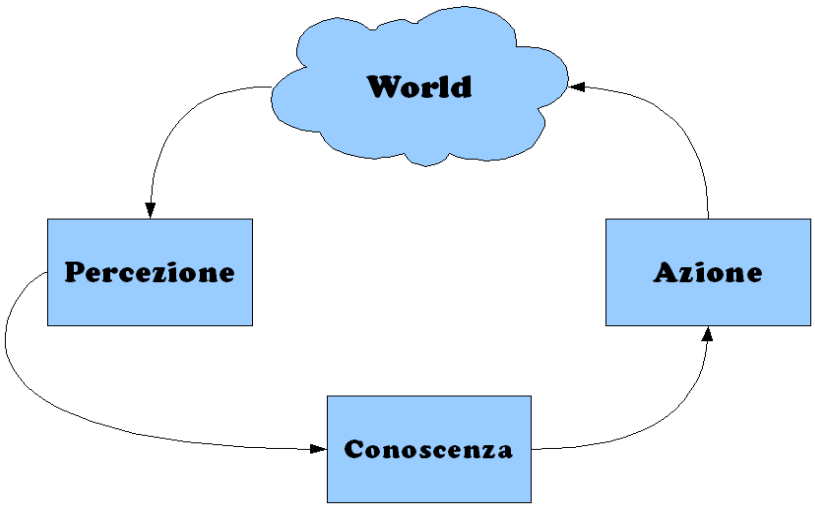

Figura 2.3: Il modello tradizionale in cui la conoscenza sta tra la percezione e l'azione

La rappresentazione del mondo viene modificata ogni volta che il robot percepisce l'ambiente e il piano delle azioni viene stabilito sulla base di tale rappresentazione.

Una delle caratterizzazioni del paradigma gerarchico è la suddivisione orizzontale e sequenziale della catena di informazioni elaborate dal sistema centralizzato (figura 2.4).

Generalmente, il modulo Pianificatore è strutturato su tre livelli:

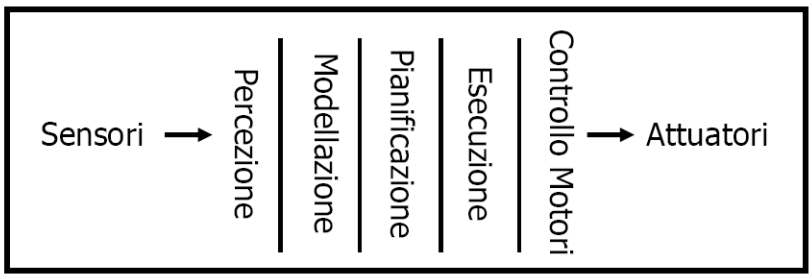

Figura 2.4: Suddivisione orizzontale del paradigma gerarchico

- Strategico
- Tattico
- Esecutivo

Il livello più alto, o strategico, genera un piano in base al compito che il robot deve eseguire. Il livello intermedio, o tattico, genera i comandi interpretando istruzioni provenienti dal livello strategico. Il livello più basso, o livello esecutivo, riceve comandi generati dal livello intermedio e si occupa del controllo in tempo reale degli attuatori.

Il vantaggio di una architettura basata su paradigma gerarchico è quello di ottenere un funzionamento predicibile, ovvero una pianificazione a priori dei comportamenti. Questo comporta, inoltre, efficienza e stabilità del sistema.

Tra gli svantaggi, si hanno l'alta complessità computazionale, dovuta principalmente alla modellazione dell'ambiente e al ragionamento, e poca adattabilità alle modifiche in tempo reale dell'ambiente e quindi si ha bassa reattività. Inoltre, la stretta sequenzialità dei moduli comporta un basso parallelismo.

Esempio di architettura gerarchica: NHC

Uno dei più importanti esempi di architettura gerarchica è il Nested Hierarchical Controller (NHC)[Mey90], schematizzato in figura 2.5. Dallo schema è subito evidente la distinzione delle tre componenti primitive:

- *SENSE*, dedicato alla ricezione degli input dai sensori, alla rappresentazione interna del mondo (world model) e al mantenimento della Knowledge Base;
- *PLAN*, dedicato alla gestione della missione e, più in particolare, alla pianificazione;

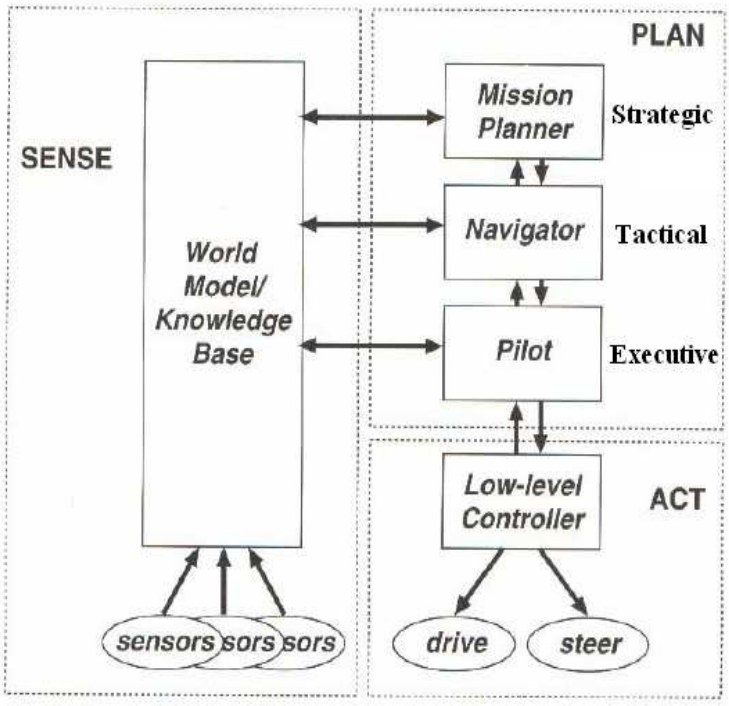

Figura 2.5: Architettura NHC

- *ACT*, dedicato alla gestione degli attuatori.

Il robot inizia a raccogliere i dati provenienti dai sensori e li combina per formare il "world model". Il modello interno del mondo contiene anche una conoscenza a priori sulle entità dell'ambiente esterno. Dopo che tale modello è stato creato (o aggiornato), il robot pianifica quali azioni deve compiere. La pianificazione, per quello che riguarda il problema della navigazione, è realizzata mediante una procedura costituita da tre passi principali, eseguiti rispettivamente dai moduli *MissionPlanner*, *Navigator* e *Pilot*, che corrispondono rispettivamente ai livelli Strategico, Tattico ed Esecutivo. Ognuno di questi moduli ha accesso alla knowledge base e al world model per completare le proprie attività, come illustrato dalle frecce di associazione di figura

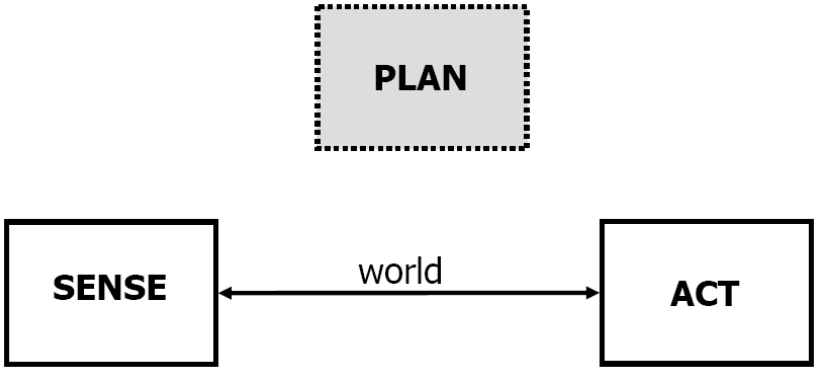

Figura 2.6: Paradigma reattivo

2.5. Ovviamente un'attività di pianificazione richiede l'interazione delle tre componenti in una sequenza ben precisa, illustrata questa volta dalle frecce di dipendenza che legano *MissionPlanner* a *Navigator* e *Navigator* a *Pilot*. Il componente *Pilot*, infine, può essere visto come interfaccia verso i controllori degli apparati fisici (motori, attuatori, servo meccanismi). Esso genera specifici comandi (primitive di tipo PLAN) da inviare al modulo LowLevel-Controller che, a sua volta, traduce questi ultimi in segnali di controllo diretti all'attuatore corrispondente.

2.1.2 Paradigma reattivo

Nel paradigma reattivo la fase di pianificazione viene completamente eliminata. Conseguentemente, si ha un netto guadagno in efficienza rispetto al paradigma gerarchico.

Nel paradigma reattivo, quindi, le funzioni primitive sono organizzate in maniera seguente:

$$SENSE \longrightarrow ACT$$

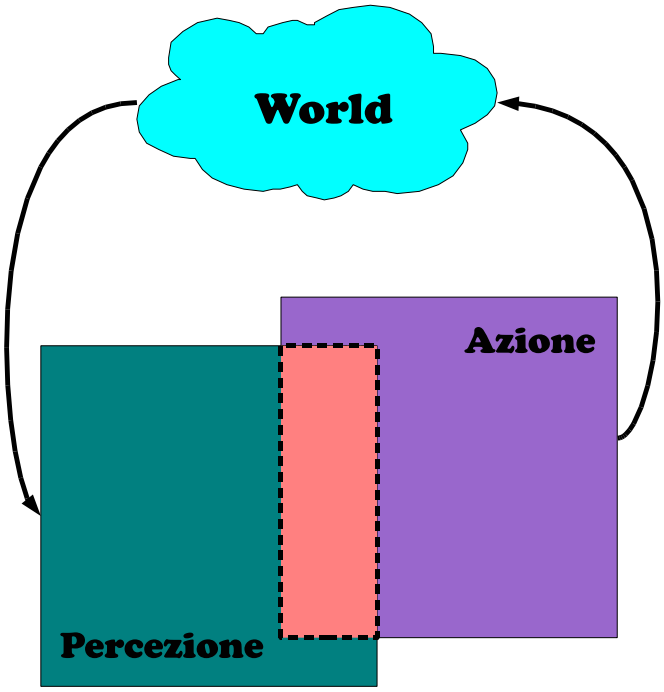

Figura 2.7: Architettura di controllo reattivo

Non esiste un modulo "Cognizione" vero e proprio e i sistemi di percezione e attuazione collaborano per definire i comportamenti del robot.

I comportamenti del robot sono reazioni alle informazioni percepite dall'ambiente. Il modulo di base di tali sistemi è costituito quindi da un comportamento (behaviour) che è ottenuto da una relazione diretta tra sensori e attuatori. Si parla, quindi, di behaviour-based robotics o sistemi reattivi, ovvero sistemi capaci di rispondere in tempo reale agli stimoli provenienti dall'ambiente circostante.

Non esiste una rappresentazione del mondo, ("The world is the best model" [Bro86]), la conoscenza del mondo non è né modellizzata, né memorizzata, ma è estratta in tempo reale dal mondo stesso attraverso i sensori. Poiché

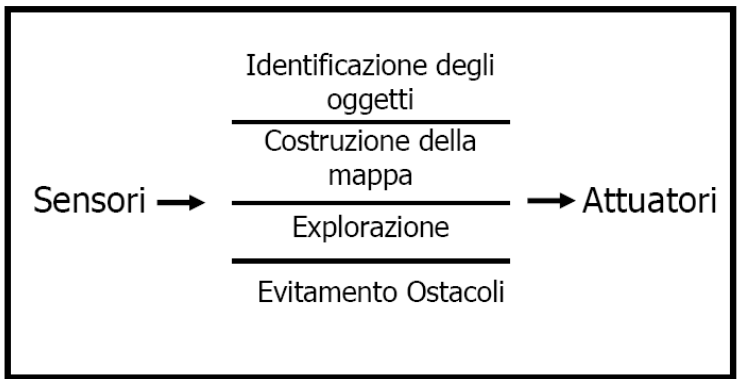

Figura 2.8: Suddivisione verticale

non esiste un modello del mondo, di conseguenza non esiste pianificazione a priori delle azioni.[Bro91a][Bro91b]

Nelle architetture di controllo reattive, l'esecuzione di un compito è suddiviso tra moduli ognuno dei quali ha assegnato una specifica competenza e attua un determinato comportamento del robot. Il comportamento complessivo è determinato dall'insieme dei comportamenti presenti.[Bro90]

Dalla suddivisione orizzontale e sequenziale del paradigma gerarchico, si passa ad una suddivisione della catena di informazioni verticale e parallela. Ogni accoppiamento di primitive $SENSE - ACT$ viene chiamato "behavior" (comportamento). Ogni comportamento sfrutta una rappresentazione locale del mondo ed elabora la migliore azione da compiere indipendentemente da altri comportamenti preesistenti e attivi.

Questa decomposizione verticale produce flussi multipli di informazione, ciascuno relativo ad una particolare funzione assegnata al robot. In questo modo, ogni sequenza si occupa di uno specifico aspetto nel funzionamento globale del sistema e può svilupparsi parallelamente ad altri processi.

Il paradigma reattivo si basa su due principi:

Principio di indipendenza. I vari moduli devono essere mutuamente indipendenti tra loro. Conseguenza immediata all'applicazione di questo principio è l'impossibilità di mantenere un modello del mondo completo, condivisibile tra tutti i moduli.

Principio di località. Ciascun sottocompito richiede, per completarsi, solo una parte limitata di tutta l'informazione sensoriale disponibile. Il robot risponde solo ad eventi del mondo senza mantenere stati persistenti: la memoria di cui ha bisogno è realizzata leggendo direttamente la situazione ambientale che gli indica il modo operativo corrente.

Riassumendo, i vantaggi del paradigma reattivo sono:

- inesistenza di un modello del mondo
- alta adattabilità alle modifiche dell'ambiente (risposte in tempo reale)
- bassa complessità di ogni livello e basso costo computazionale complessivo del sistema
- possibilità di avere parallelismo nel controllo
- estensione dei comportamenti relativamente semplice

Mentre gli svantaggi:

- difficoltà nel prevedere a priori il comportamento globale del robot
- gestione della concorrenza tra moduli
- aumentando i comportamenti si aumenta anche la complessità della gestione della concorrenza tra moduli con conseguente difficoltà nella risoluzione di conflitti

Un esempio di architettura reattiva: architettura subsumption

Si tratta di un sistema di controllo per la navigazione senza collisioni di un robot mobile dotato di sensori ad ultrasuoni. L'ideatore di questo tipo particolare di architettura è R. A. Brooks, [Bro99].

Nelle architetture subsumption, l'esecuzione di un compito è suddiviso tra moduli, ognuno dei quali ha assegnato una specifica competenza e attua un determinato comportamento del robot. I comportamenti sono organizzati in una architettura a strati ed hanno priorità diversa a seconda del loro livello. Comportamenti a livello più alto (high priority) possono inibire le operazioni o sopprimere l'input dei comportamenti a livello più basso (low priority).

Ogni modulo ha delle linee di output e di input. Linee di output di un modulo possono essere collegate a linee di input o di output di altri moduli.

In questo tipo di architettura si evidenziano quindi vari livelli di competenza:

- Livello 0: evitare gli ostacoli
- Livello 1: navigazione casuale nell'ambiente
- Livello 2: esplorazione del mondo e "identificazione" dei punti di interesse
- Livello 3: costruzione della mappa dell'ambiente (relazioni tra punti)
- Livello 4: rilevazione dei cambiamenti dell'ambiente statico
- Livello 5: ragionamento sul mondo in termini di esecuzione di task in relazione al rilevamento di determinati oggetti
- Livello 6: formulazione ed esecuzione di comportamenti che determinano cambiamenti dello stato del mondo

- Livello 7: ragionamento sui comportamenti degli oggetti e modifica dei piani in accordo

Ogni strato dell'architettura è implementato con una macchina a stati finiti in grado di inviare messaggi alle altre e di gestire proprie strutture dati. Non esiste memoria globale condivisa. Ogni macchina agisce in maniera asincrona rispetto alle altre, monitorando i propri input e inviando messaggi di output. Input ai moduli possono essere soppressi e output inibiti dagli output di altri moduli (meccanismo tramite il quale strati superiori assumono il controllo degli stati inferiori).

2.1.3 Architetture ibride

Nonostante il paradigma reattivo abbia diverse proprietà desiderabili, tra cui l'efficienza di esecuzione, la tendenza oggi è quella di preferire architetture ibride di tipo deliberativo-reattivo. In queste architetture, il robot per prima cosa pianifica (da cui il termine di deliberativo) il modo migliore di decomporre un task (una missione) in subtask (sotto-missioni). Dopo questa fase di pianificazione, il robot decide quali siano i comportamenti (behavior) da assumere per affrontare una missione e, di conseguenza, attiva una fase di tipo $SENSING - ACTING$. La fase di pianificazione viene eseguita in un singolo passo, mentre la fase di $SENSING - ACTING$ combina in un solo passo primitive SENSE con primitive ACT, come illustrato in figura 2.9. L'organizzazione sensoriale in questo paradigma, come si può intuire, è una fusione degli stili gerarchico e reattivo.

Nel paradigma ibrido, le funzioni primitive sono organizzate nel seguente modo:

$$PLAN, SENSE \longrightarrow ACT$$

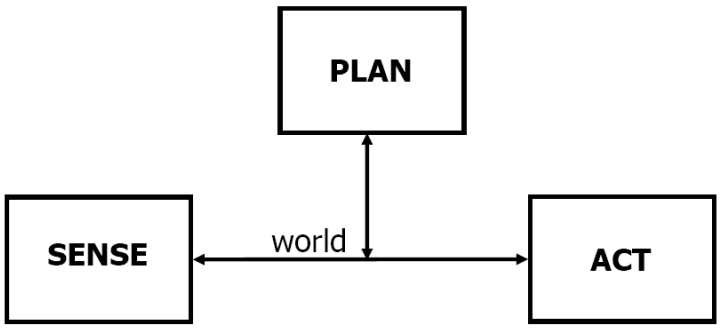

Figura 2.9: Paradigma ibrido

Un approccio puramente reattivo dota il robot della capacità di eseguire compiti semplici in maniera efficiente, come ad esempio evitare gli ostacoli e adattarsi alle variazioni del mondo, ma non garantisce l'esecuzione di task più complessi, che includono modellizzazione dell'ambiente e comportamenti più complessi. L'integrazione dei metodi reattivi con i metodi gerarchici combina l'efficienza della pianificazione con la flessibilità dei sistemi di controllo reattivo.

Tipicamente, una architettura ibrida comprende un modulo pianificatore strategico e un modulo pianificatore tattico per la gestione dei comportamenti di un robot. Il pianificatore strategico pianifica a lungo termine le azioni del robot, individuando la sequenza di sotto-obiettivi da realizzare per raggiungere il goal e passando i risultati per l'esecuzione al pianificatore tattico. Il pianificatore tattico inizializza e monitora i comportamenti prendendosi cura degli aspetti temporali per la loro coordinazione.

2.1.4 Architetture decentralizzate o distribuite

Un approccio alternativo all'uso di un unico sistema robotico per l'esecuzione di compiti è l'uso di un gruppo di sistemi robotici, ciascuno di complessità

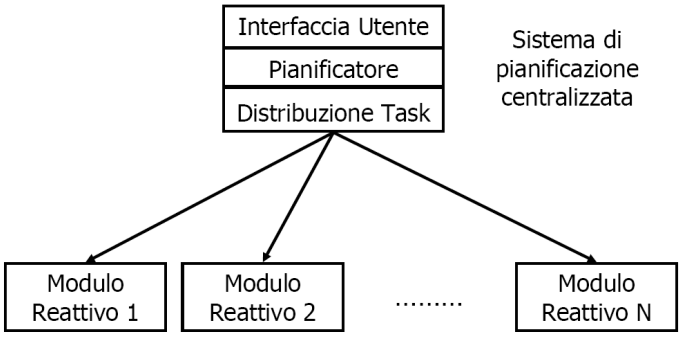

Figura 2.10: Schema generale di una architettura ibrida

inferiore, che cooperano per l'esecuzione dello stesso compito. L'intelligenza è distribuita tra i vari sistemi, ognuno dei quali è autonomo.

2.2 Un approccio differente: la percezione attesa

Le architetture precedenti hanno dimostrato nel tempo i loro limiti nelle applicazioni del "mondo reale", nelle quali sono richieste reattività e autonomia. Il mondo reale solitamente è un ambiente dinamico e ricco di elementi che cambiano nel tempo e che possono seriamente influire sulla performance di un robot. Le architetture reattive si basano puramente su feedback sensoriale: se ne possono trovare esempi in [Ark98][Bro86]. In esse, il concetto di mondo è ridotto a un modello interamente non strutturato e impredicibile. Nell'approccio anticipatorio, il mondo è ugualmente considerato sia dinamico che ricco di potenziali elementi di disturbo, ma non è visto come un modello completamente non strutturato, caotico e imprevedibile. In pratica, il mondo è almeno parzialmente strutturato e quindi predicibile. Un robot può quindi

sfruttare questa caratteristica per evitare inutili processi percettivi attraverso meccanismi anticipatori.

Nel corso degli anni, le architetture robotiche hanno visto affermarsi con sempre maggiore diffusione il paradigma anticipatorio. Esso si basa sul principio che dotare i sistemi robotici della capacità di formulare aspettative sulle conseguenze sensoriali dei propri atti, costituisca un vantaggio nel controllo motorio, nel corso dell'esecuzione di un task più o meno complesso. Ciò è coerente con l'evidenza, proveniente da studi neuroscientifici [WK98], che il controllo motorio, nell'Uomo, si basi non solo sul feedback sensoriale ma anche su predizioni di tale ingresso sensoriale [CTP06]. Si pone, allora, l'esigenza di uno strumento utile a rendere un sistema in grado di formulare predizioni. Uno strumento simile è offerto dai modelli interni diretti. I modelli diretti, adoperati usualmente sia nella teoria del controllo sia nelle neuroscienze, sono pensabili, in astratto, come sistemi che ricevono in ingresso dati riguardanti lo stato interno corrente dell'agente e i comandi motori ricevuti, e forniscono in uscita lo stato sensoriale futuro, quindi, appunto, un'aspettativa sensoriale, o "Percezione Attesa" (Expected Perception - EP).

2.2.1 Gli studi nelle neuroscienze

Recenti studi in neuroscienze [Ber98][DG00] [CC], soprattutto sulla parte del controllo del movimento nell'uomo, hanno suggerito un comportamento anticipatorio del cervello, in relazione almeno ad alcuni compiti. Per controllo motorio si intende la capacità del sistema nervoso di regolare o dirigere il movimento. Quest'ultimo è spesso descritto nel contesto dell'esecuzione di una particolare azione, tipo camminare, correre, il controllo della stazione eretta. In pratica si studia come viene controllato il movimento nel contesto di una specifica attività, assumendo che ciò fornisca informazioni sui principi

generali. Il movimento, poi, non può essere studiato senza tenere conto della percezione. Per percezione si intende l'integrazione dei dati grezzi sensoriali in informazioni fruibili cognitivamente. E' importante sottolineare che percezione e azione sono essenziali l'una all'altra: la percezione, cioè, non è un meccanismo passivo, ma attivo, che mira ad anticipare le conseguenze sensoriali di un'azione, legando quindi le informazioni sensoriali e i comandi motori in un tessuto computazionale coerente. I sistemi sensoriali/percettivi forniscono informazioni sullo stato del corpo (ad esempio, la posizione del corpo nello spazio) e sulle caratteristiche dell'ambiente, critiche per la regolazione del movimento (ad esempio, la presenza di ostacoli). Le informazioni provenienti dalla periferia somatica, da un lato innescano una serie di risposte riflesse che svolgono cruciali funzioni protettive e posturali, dall'altro, forniscono ai centri superiori un continuo aggiornamento sulla posizione degli arti, del corpo e sul grado di contrazione dei muscoli, indispensabile per l'esecuzione di una fine attività motoria. Per poter adeguare in modo ottimale i propri comportamenti motori alle mutevoli ed imprevedibili richieste ambientali, il Sistema Nervoso Centrale necessita, infatti, di un flusso costante di informazioni sensoriali. Agli esterocettori cutanei ed ai telecettori (vista, udito, olfatto) spetta il compito di fornire un dettagliato sistema di coordinate visuo-spaziali all'interno delle quali si esprimerà il movimento, mentre i propriocettori dei muscoli e delle articolazioni, unitamente al sistema vestibolare, forniscono continui ragguagli sullo stato del sistema effettore. L'utilizzo delle informazioni sensoriali per gli aggiustamenti motori avviene secondo due meccanismi fondamentali definiti, rispettivamente, a *feed-back* ed a *feed-forward* [KSJ00]. Il meccanismo a *feed-back* implica la presenza di un sistema comparatore che, previo confronto tra il valore di un parametro in uscita ed il valore di riferimento desiderato, agisce su un sistema opera-

tore per modificare il valore del parametro in uscita fino a farlo coincidere con il valore desiderato. Nell'ambito del sistema motorio, il meccanismo a *feed-back* interviene in tutti quei casi in cui bisogna mantenere una variabile (la posizione di un'articolazione o la forza di un muscolo) ad un valore prestabilito come, ad esempio, nel mantenimento della postura e nel controllo dei movimenti lenti. Se ne deduce che il sistema a *feed-back* richiede tempo per poter analizzare i vari segnali, per cui è inefficace quando le variabili ambientali mutano troppo velocemente (si pensi al gesto di afferrare una palla in corsa). In questi casi le informazioni sensoriali devono essere utilizzate per programmare il movimento corretto prima che gli eventi che innescano il movimento stesso si realizzino (nell'esempio della palla, prima che essa attraversi lo spazio di afferramento); si parla, quindi, di controllo anticipatorio od a *feed-forward*. Nel meccanismo a *feed-forward*, le informazioni sensoriali condizionano il programma motorio prevedendo una data successione di eventi (traiettoria e velocità della palla, per restare in argomento); ne consegue che tutto funziona fino a quando gli eventi si realizzano secondo la sequenza prevista, ma non quando interviene una variazione imprevista. Meccanismi a *feed-back* ed a *feed-forward* non sono mutuamente esclusivi, anzi, intervengono spesso in successione per assicurare la perfetta riuscita dell'atto motorio (dopo aver afferrato la palla grazie al meccanismo a feed-forward, si riesce a mantenere la presa grazie agli aggiustamenti della forza resi possibili dai meccanismi a *feed-back*).[Per99]

Uno schema di controllo *feedforward*, quindi anticipativo, è basato sulla computazione preprogrammata delle forze che saranno necessarie al sistema nell'esecuzione di un determinato piano motorio, senza l'utilizzo di informazioni sensoriali se non per la formulazione dello stato desiderato. Per fare questo è necessario che il sistema nervoso mantenga una rappresentazione

interna o modello inverso della fisica del corpo e degli attuatori. Un tale meccanismo di controllo è veloce e non corre il rischio dell'instabilità, ma ha un ovvio inconveniente: la sensibilità a disturbi imprevisti, a modificazioni del corpo (per esempio la presenza di utensili) e/o degli attuatori (per esempio, l'affaticamento). Ciò suggerisce che un controllore *feedforward* debba necessariamente avere caratteristiche adattative [KG92][WK98].

Diversi studi hanno mostrato che il sistema nervoso utilizza rappresentazioni interne per anticipare le conseguenze delle forze di interazione dinamica. Questi studi nel complesso suggeriscono che il sistema nervoso ha capacità sofisticate di controllo anticipativo, per cui sono necessari modelli interni accurati della dinamica del corpo e delle interazioni con l'ambiente. Si è detto che un inconveniente della modalità di controllo *feedforward* è quello di non essere in grado di rispondere a perturbazioni inaspettate. Se queste possono essere misurate, o se più in generale, sono disponibili informazioni sensoriali sull'evoluzione del movimento, queste possono essere utilizzate per effettuare correzioni in tempo reale del movimento stesso, in modo che questo segua il piano motorio desiderato. Ciò corrisponde alla modalità di controllo a *feedback*. Il problema è che l'informazione sensoriale è disponibile con un ritardo non trascurabile e questo pone seri problemi per la stabilità del controllo.

Non ci sono prove dirette ed inequivocabili che il sistema nervoso utilizzi meccanismi a *feedforward* basati su modelli interni che stimano lo stato presente sulla base di una informazione sensoriale passata, ma ci sono numerose evidenze indirette: per esempio è stato mostrato [DG00] che in situazioni di raggiungimento di un target in movimento la prima evidenza di correzione della traiettoria risale a non più di 110 millisecondi dopo lo spostamento, un intervallo incompatibile con i ritardi sensoriali che sono molto maggiori.[CC]

Nel controllo motorio, infatti, ci sono ritardi nei trasduttori, nei calcoli

principali e nell'output motorio. Le latenze nei sensori di trasduzione sono più rilevanti nel sistema visivo, dove la retina introduce un ritardo di 30-60 millisecondi e i ritardi di trasmissione possono essere apprezzabili. Sono presenti inoltre ritardi centrali dovuti ad eventi non ancora molto chiari, come la computazione neuronale, il compiere delle scelte e ai colli di bottiglia nell'esecuzione dei comandi. Ci sono ritardi nell'output motorio derivanti dagli stessi ritardi di trasmissione degli assoni motoneuronali, ritardi di eccitazione-contrazione dei muscoli e ritardi di fase dovuti all'inerzia del sistema. Tutti questi ritardi portano ad un inevitabile ritardo di *feedback* che va dai 30 millisecondi per un riflesso a livello del midollo spinale ai 200-300 millisecondi per un movimento guidato dalla visione.

Movimenti delle braccia veloci e coordinati non possono essere eseguiti sotto un puro controllo di *feedback*, e quindi correttivi, dato che i cicli *feedback* biologici sono troppo lenti e hanno poco guadagno. [Kaw99]

2.2.2 Il paradigma anticipatorio

In robotica, il problema dei ritardi senso-motori è analogo al problema dei ritardi nei modelli *feedback* neurofisiologici. E' interessante sottolineare le connessioni e il parallelismo che esiste tra la robotica e le neuroscienze, si veda a tal proposito la figura 2.11[DTL+03b]. Entrambe devono far fronte agli stessi tipi di problemi, come quello dei ritardi di cui si è parlato precedentemente. Nella robotica, per ottenere una coordinazione senso-motoria con prestazioni paragonabili a quelle "naturali", vengono emulati i modelli che si riscontrano nelle neuroscienze, come per esempio i cicli *feedback*. Per tale ragione, la robotica cerca continua ispirazione negli studi e nelle scoperte delle neuroscienze. Dallo studio dei modelli di coordinazione senso motoria dell'uomo, per ottenere la stessa capacità adattiva nei confronti dell'ambiente

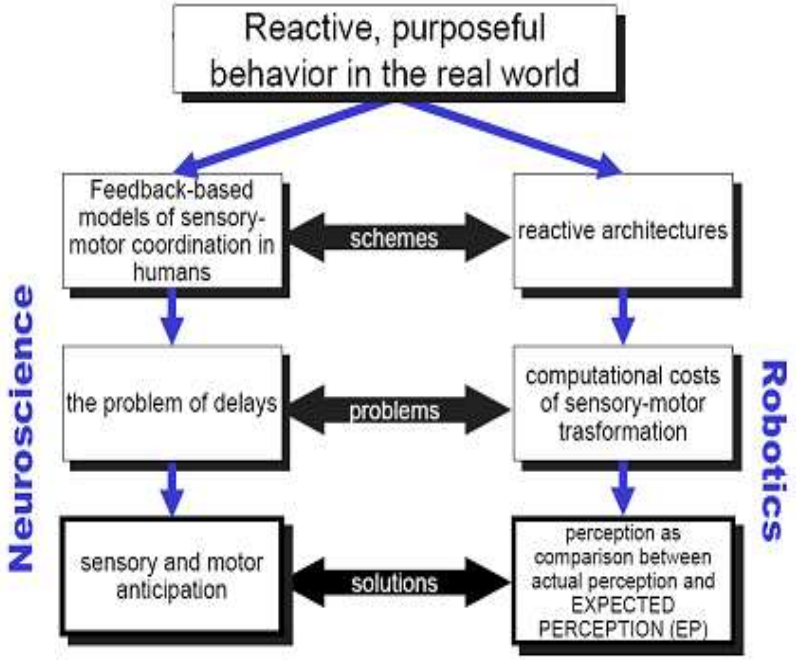

Figura 2.11: Parallelismo tra Neuroscienze e Robotica

in cui vive, sono state sviluppate le architetture reattive per le applicazioni robotiche. Se nell'uomo abbiamo il problema dei ritardi, in maniera speculare, nelle macchine questi ritardi sono dovuti ai costi computazionali delle trasformazioni senso-motorie(fig.2.11). La soluzione di questo problema nell'uomo viene risolta considerando la percezione come confronto tra valori predetti e dati sensoriali di feedback. Analogamente la robotica considera la percezione come confronto tra la percezione attuale e la percezione attesa (**Expected Perception**), adottando il paradigma anticipatorio.

Possiamo distinguere due fasi di anticipazione:

anticipazione sensoriale Capacità di generare previsioni su quello che sarà

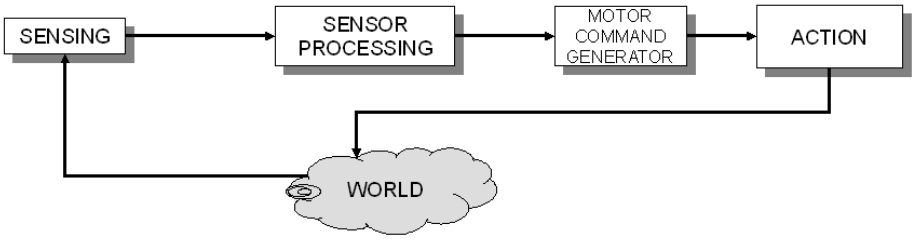

Figura 2.12: Schema a feedback

l'ingresso sensoriale del sistema in istanti futuri;

anticipazione motoria Capacità di eseguire azioni preparatorie.

L'anticipazione sensoriale permette di ottenere cicli di coordinazione sensomotoria più veloci, nel caso di dati sensoriali complessi, e maggiore accuratezza, nel caso di incertezza (rumore) dei dati sensoriali.

L'anticipazione motoria, d'altro canto, permette di realizzare una più efficiente manipolazione e interazione con l'ambiente (mondo).

Risulta chiaro come sia molto importante il fattore reattività, inteso come capacità di reagire in tempo reale agli eventi del mondo che possono disturbare l'esecuzione di un certo compito. I modelli a *feedback* non sono sempre adatti per modellare questa capacità, specialmente quando si ha a che fare con dati sensoriali molto complessi, a causa appunto del problema dei ritardi.

In Figura 2.12 è mostrato lo schema a *feedback* classico, in cui i sensori acquisiscono i dati, che vengono successivamente processati ed, in base ai valori ottenuti, si provvede a generare i comandi motori opportuni. Questi sono poi inviati agli attuatori che eseguono il movimento. L'azione eseguita si ripercuote sull'ambiente (il mondo) e il ciclo si ripete; in tal modo il sistema reagisce a seconda dei cambiamenti avvenuti, siano essi dipendenti o indipendenti dalle azioni compiute.

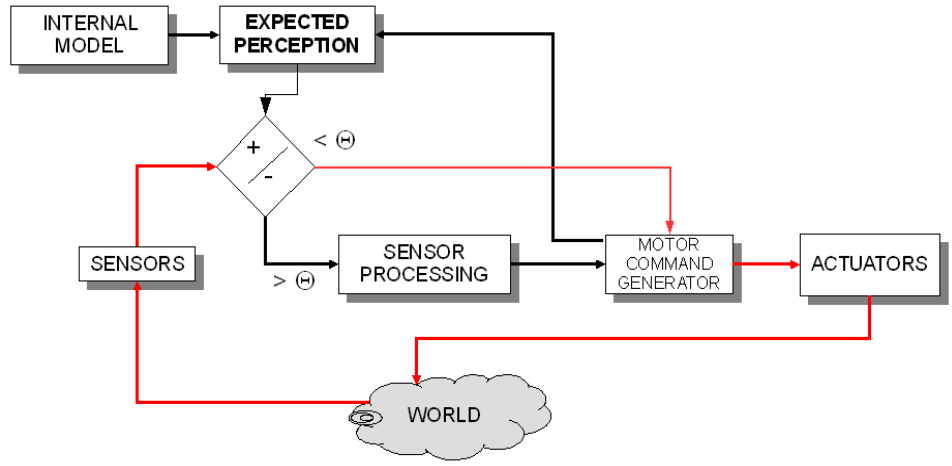

Figura 2.13: Schema Expected Perception

In Figura 2.13 è mostrato lo schema del paradigma anticipatorio. La principale differenza consiste nell'introduzione du un nuovo modulo, quello dell' **Expected Perception**. Quest'ultimo, tramite uno o più modelli interni, attua una previsione dei dati sensoriali grezzi, cioè privi di qualunque processing. Questi vengono quindi confrontati con quelli reali. Se dal confronto risulta che la differenza tra i dati previsti e quelli reali sono minori di una certa soglia Θ, allora si conclude che la previsione è corretta e si passa direttamente alla generazione dei comandi motori, evitando in tal modo l'intera fase di processing dei dati. In caso contrario, significa che c'è stato qualche imprevisto e si esegue il loop classico della Figura 2.12. In definitiva, se si ha successo nella previsione, si ha un risparmio computazionale pari all'intera procedura di analisi dei dati sensoriali, altrimenti si attuano le stesse operazioni che si adotterebbero con l'architettura tradizionale.[DTL+03a]

Il vantaggio principale nell'usare il paradigma anticipatorio è quindi ora più evidente, in caso di complesse e costose operazioni di processing dei dati,

il risparmio è notevole.

L'uso di un confrontatore con soglia porta ad un altro grande vantaggio: quello di una maggiore affidabilità dei dati nel caso in cui essi siano affetti da incertezza e/o rumorosità. E' necessario però un modello interno del mondo che permetta di effettuare le predizioni.

Capitolo 3

Sistemi di navigazione autonoma di robot mobili

La navigazione autonoma resta una delle sfide più interessanti in ambito robotico, anche perché coinvolge molti aspetti dell'intelligenza artificiale: la percezione, l'azione, la pianificazione, le architetture, l'hardware, l'efficienza computazionale e la risoluzione di problemi.

Il problema principale riguarda la pianificazione della sequenza di azioni necessaria a raggiungere una ben determinata posizione. Un'altro problema da affrontare è quello della costruzione della mappa, che introduce il problema di come il robot possa accuratamente localizzare se stesso durante il suo movimento. [Mur00]

Gli obiettivi della navigazione possono essere espressi attraverso le seguenti quatto domande:

Dove sto andando? Di solito è stabilito dall'operatore umano o da un pianificatore, è il goal da raggiungere.

Qual'è il miglior percorso? E' il problema della pianificazione ed è il set-

tore della navigazione di maggior interesse.

Dove sono stato? La costruzione di una mappa è un aspetto da non sottovalutare nella navigazione. Sia quando un robot deve esplorare nuovi ambienti, che quando si trova ad agire sempre nello stesso ambiente; avere una mappa potrebbe migliorare la sua performance.

Dove sono? Per poter seguire un percorso o costruire una mappa, il robot deve sapere dove si trova; ci si riferisce a questo problema come al problema della localizzazione.

Per rispondere a questi quesiti il nostro sistema deve essere composto da:

- un algoritmo che permette l'esplorazione dell'ambiente;
- una struttura dati per la rappresentazione del mondo, in cui vengono memorizzati gli ostacoli incontrati durante l'esplorazione; questa struttura dati è una mappa dell'ambiente e può essere di due tipi: qualitativa, se espressa tramite dei landmark e dei gateway, o quantitativa, se espressa tramite le distanze tra gli ostacoli;
- un algoritmo di path planning, che permette di scegliere il percorso migliore dalla posizione di partenza fino all'obiettivo;
- un algoritmo di localizzazione che permette di localizzare il robot all'interno della mappa e quindi di proseguire il percorso stabilito;
- un algoritmo di obstacle avoidance che permette di evitare gli ostacoli del mondo reale.

Nei paragrafi successivi, gli argomenti dei punti precedenti verranno approfonditi; una trattazione completa dell'argomento può essere trovata in [Lat91].

3.1 Esplorazione

La costruzione della mappa del mondo dove il robot andrà ad operare deve essere eseguita esplorando il mondo stesso. Per far questo, bisogna usare un algoritmo di esplorazione affinché il robot analizzi l'ambiente circostante e possa identificarne gli ostacoli e i punti di riferimento (milestone). Tale procedura può essere eseguita in vari modi:

- vagando nell'ambiente in maniera random; statisticamente dopo un lungo periodo si è coperta l'intera area
- usare la propriocezione (tramite odometria) per non tornare ad esplorare parti dell'area già viste e quindi esplorarne delle nuove
- usare il centroide di aree sconosciute della mappa come punto d'arrivo del proprio percorso; andando avanti si riduce la zona sconosciuta e di volta in volta si individua un nuovo goal. Questo approccio è di facile implementazione ma è inefficiente nel caso di più aree inesplorate

Ci sono due algoritmi che seguono quest'ultimo metodo e che si differenziano nel modo in cui individuano il nuovo obiettivo: Frontier-based exploration e Generalized Voronoi Graph.

3.1.1 Frontier-based exploration

Per ogni area si può supporre che ci sia un confine rappresentato da ostacoli e parti sconosciute. [Mur00]

Ogni linea della frontiera deve essere esplorata e la scelta può essere fatta principalmente in due modi:

- esplorare prima la frontiera più vicina

- esplorare prima la frontiera più grande

Nel secondo caso, per la valutazione della frontiera, si usa come unità di misura il lato delle celle della *griglia di occupazione* (Occupancy Grid), che è una struttura dati utilizzabile per rappresentare il mondo reale, necessaria per questo algoritmo.

Se un lato della cella di frontiera è in contatto con un altro lato di una cella di frontiera, questi due lati si connettono formando una linea. Per eliminare l'effetto del rumore dei sensori si considera frontiera una linea che contiene un numero di celle superiore ad una data soglia.

Una volta stabilita la frontiera bisogna individuare il punto d'arrivo (goal) del robot. Un buon obiettivo è il centroide della frontiera che altro non è che il punto in cui si trova la x media e la y media delle celle della frontiera.

A questo punto, basta che il robot si muova verso il goal stabilendo il percorso, evitando gli ostacoli che vi si trovano e aggiornando la mappa, le frontiere e l'obiettivo.

3.1.2 Generalized Voronoi graph

Un altro modo per esplorare la zona di interesse è far costruire dal robot un grafo di Voronoi generalizzato ridotto.[CB00]

Questo metodo costruisce un percorso che tenta di rimanere equidistante da tutti gli oggetti individuati e che è un lato di un Generalized Voronoi Graph (GVG) (per ulteriori dettagli si veda sec. 3.2.1).

Il robot segue il percorso fino ad incontrare un gateway o finché non termina. In quei punti, il robot si trova a scegliere uno dei lati del grafo che da lì si diramano. Nel caso in cui nessuno di questi lati sembra andar bene perché terminano con oggetti, l'algoritmo fa backtracking lungo il path tornando all'inizio o ad un altro ramo. Se il robot incontra un ramo alternativo durante

il suo cammino su un lato, può scegliere in maniera random quale seguire e mantenersi in memoria gli altri che potranno servire in caso di backtracking. L'algoritmo termina quando è stata esplorata tutta l'area.

3.2 Rappresentazione del mondo

Un robot, per muoversi in un ambiente, generalmente, ha bisogno di una mappa dove memorizzare gli oggetti "percepiti" e sapere dove si trova rispetto a questi (localizzazione).

La configurazione spaziale (Cspace) è una struttura dati che permette ai robot di specificare la posizione di ogni oggetto e di sé stesso.

Le coordinate degli oggetti possono essere espresse con 2 o 3 dimensioni a seconda che ci interessi ricordare l'altezza oppure no (nel caso che il robot poggi a terra e non debba salire sopra agli oggetti). L'orientazione del robot rispetto alla mappa può essere similmente rappresentata da 1,2 o 3 angoli di Eulero.[Mur00]

3.2.1 Rappresentazioni Cspace

Tra le rappresentazioni Cspace più diffuse si trovano le Meadow Map, i diagrammi di Voronoi, le griglie regolari (Occupancy Grid) e i Quadtree. Queste rappresentazioni si differenziano per la maniera in cui suddividono lo spazio libero. È considerato libero un qualsiasi spazio aperto in cui non vi è alcun oggetto che il robot può urtare.

Meadow Map

Le Meadow Map (MM) trasformano lo spazio libero in poligoni convessi.[Mur00]

I poligoni hanno una proprietà importante: se il robot parte da un punto sul perimetro e prosegue su una linea dritta verso un altro punto del perimetro, non uscirà dal poligono. Quindi il poligono rappresenta una regione sicura che il robot può attraversare.

In questo modo, la pianificazione diventa la risoluzione della tracciatura della miglior serie di poligoni da attraversare.

L'algoritmo considera gli oggetti del mondo dello stesso ordine di grandezza del robot e quest'ultimo lo presume olonomico (può ruotare su sé stesso). Il primo passo è la costruzione dei poligoni convessi da considerare segmenti lineari tra coppie di oggetti interessanti (di riferimento). L'algoritmo, quindi, suddivide lo spazio libero in poligoni complessi che usa come segmenti lineari.

I problemi che presenta tale struttura dati sono i seguenti:

- alcuni segmenti lineari che formano il perimetro non sono connessi ad altri poligoni e dovrebbero essere trascurati dall'algoritmo di pianificazione
- alcuni segmenti lineari sono lunghi e l'algoritmo di pianificazione dovrebbe considerare la possibilità di tagliare attraverso il poligono; non è facile per un computer però discretizzare un segmento lineare.

Generalized Voronoi Graph

Un Generalized Voronoi Graph (GVG) può essere costruito non appena un robot entra in un nuovo ambiente [CB00]. L'idea di base di un GVG è generare una linea, chiamata Voronoi Edge (VE), equidistante da tutti i punti. Come mostrato in figura 3.1, questa linea passa nel mezzo del corridoio e delle aperture. Il punto dove molti VE si incontrano è conosciuto come Voronoi vertex (VV) e spesso ha una corrispondenza fisica con la configura-

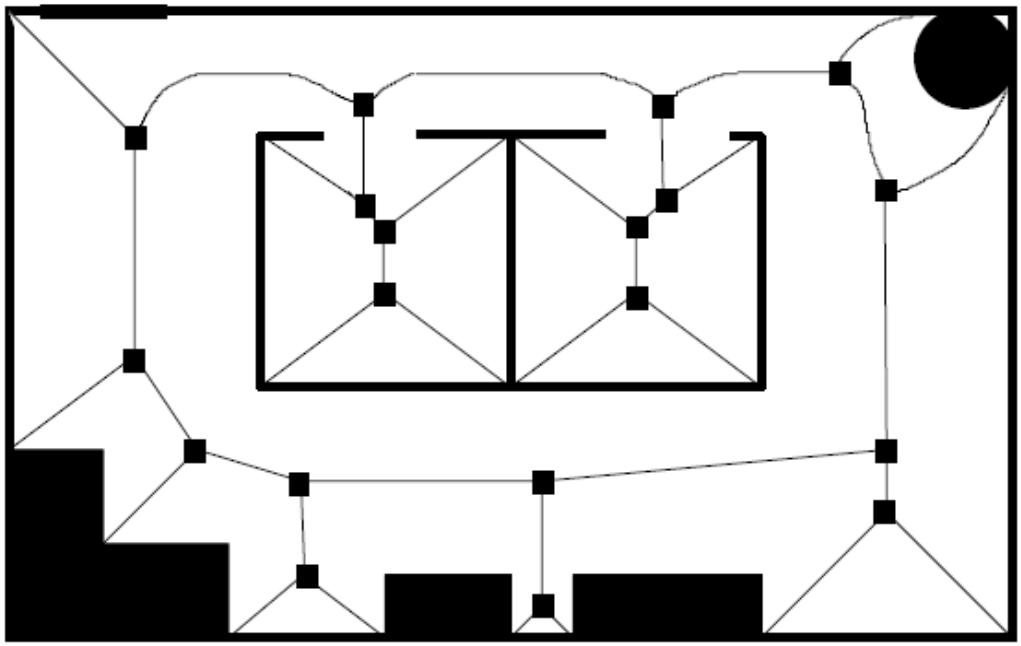

Figura 3.1: Grafo da un GVG

zione dell'ambiente osservato. Con questo tipo di struttura dati il robot può semplicemente seguire i VE stando così il più lontano possibile dagli ostacoli.

Il problema con questa struttura dati è che non può essere modellata con un grafo perché è costituita da linee curve che non possono essere trasformate in archi dei grafi. Questo è un problema perché durante la pianificazione del percorso, generalmente, si trasforma la mappa in un grafo per scegliere il percorso ottimale.

Griglie Regolari

Il metodo delle griglie regolari (GR) sovrappone una griglia cartesiana 2D sullo spazio del mondo come in figura 3.2 [Mur00]. Se c'è un oggetto nell'area contenuta nell'elemento della griglia, questo elemento è marchiato come occupato.

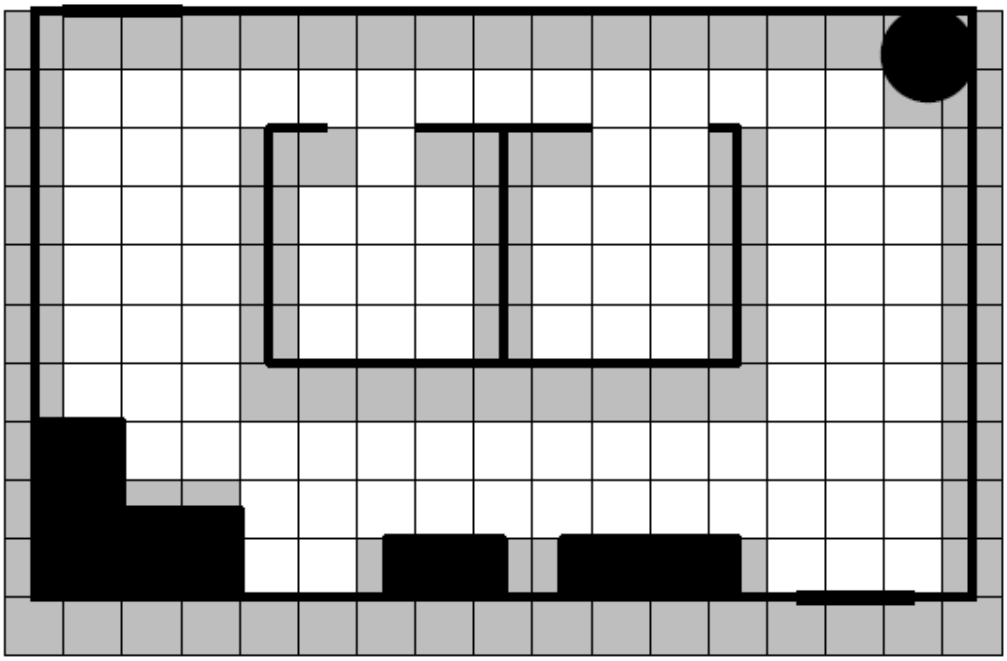

Figura 3.2: Griglia Regolare

Questa struttura dati può essere facilmente trasformata in un grafo in cui i centri degli elementi della griglia diventano i nodi del grafo e gli archi collegano gli elementi vicini. I grafi originati da Griglie Regolari risultano essere altamente connessi (ogni nodo può avere 4 o 8 archi a seconda se si considerano o meno vicini anche i nodi sulle diagonali).

Il problema delle griglie regolari è che introducono la polarizzazione digitale, cioè se un oggetto si trova in una piccola porzione dell'elemento, tutto l'elemento viene marchiato come occupato producendo oggetti frastagliati e perdita inutile di spazio libero. Una soluzione è ridurre la grana della griglia, ma questo porta ad una maggior occupazione di memoria e ad un maggior numero di nodi da elaborare da parte dell'algoritmo di pianificazione. E' comunque la struttura dati più utilizzata in quanto, ad una ridotta complessità, aggiunge una facile manipolazione da parte degli altri algoritmi.[BMF]

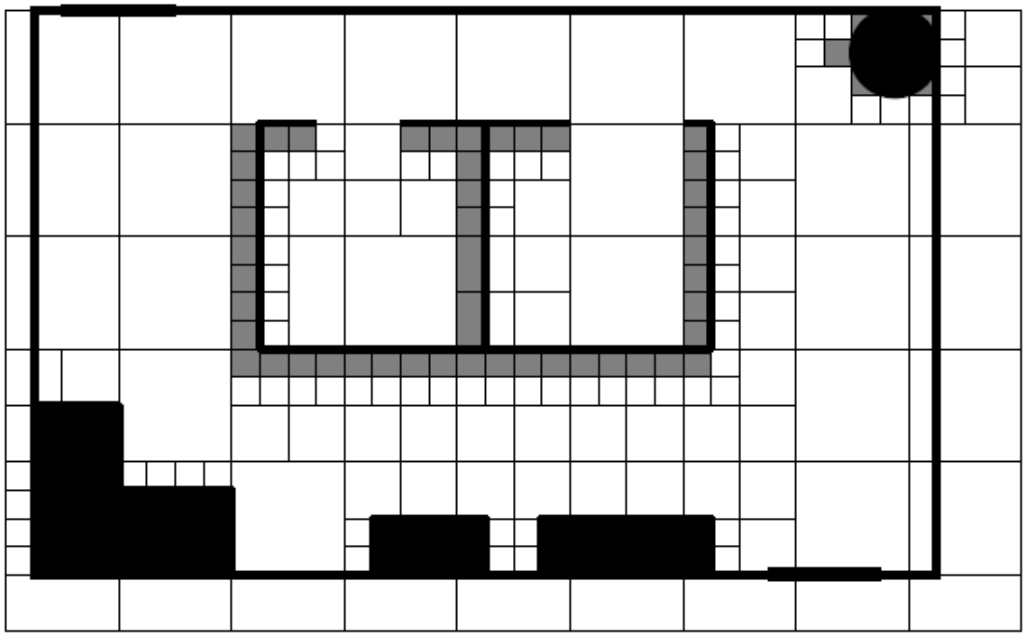

Figura 3.3: Rappresentazione di un Quadtree

Quadtrees

Sono una variante delle griglie regolari che permette di evitare la perdita inutile di spazio libero.[Mur00]

Il Quadtree è una griglia ricorsiva:

- la rappresentazione parte con elementi della griglia piuttosto larghi
- se un oggetto cade in una parte della griglia, ma non in tutta, l'algoritmo divide l'elemento in 4 elementi della griglia più piccoli (da qui il nome "quad")
- se l'oggetto non copre un particolare sotto-elemento, l'algoritmo esegue altre divisioni ricorsive di questo elemento in 4 sotto-elementi
- questo procedimento può avere una risoluzione finale, cioè si può de-

cidere la dimensione minima oltre la quale non si può dividere più in sotto-elementi

Purtroppo i percorsi generati dal Quadtree sono sotto-ottimali, perché i segmenti devono passare per forza dal centro delle celle.

Framed-Quadtrees Per risolvere il problema del Quadtree classico, si può usare questa struttura dati che si differenzia dalla precedente per l'aggiunta, intorno al perimetro di ogni regione del Quadtree, di celle ad alta risoluzione. Nella figura 3.4 sono a confronto 3 percorsi generati con Griglie Regolari, Quadtrees e Framed-Quadtrees. In quest'ultimo percorso, sono presenti piccoli rettangoli grigi intorno alle celle che sono le celle di bordo di ogni quadrante. Questa rappresentazione permette molti angoli di direzione, invece del massimo di 8 come nel caso delle griglie regolari. Un tratto del percorso può essere costruito tra 2 celle di bordo (border cells) e approssima i percorsi ottimali.[YSSB98]

L'inconveniente di questa struttura dati è che può occupare molta memoria rispetto alle griglie regolari, soprattutto per ambienti molto grandi.

3.3 Costruzione della mappa

Dopo aver scelto la struttura dati che rappresenterà il mondo nel quale il robot dovrà muoversi, bisogna trovare un algoritmo per riempire la struttura dati a meno che non lo si voglia fare a mano. Un metodo automatico è, invece, quello di far esplorare il mondo al robot stesso e prendere i dati dai suoi sensori per trasferirli nella struttura dati. Per fare questo "trasferimento" abbiamo bisogno di un modello dei sensori che permetta di capire come i dati ricevuti descrivano il mondo reale e di un algoritmo che riporti tali

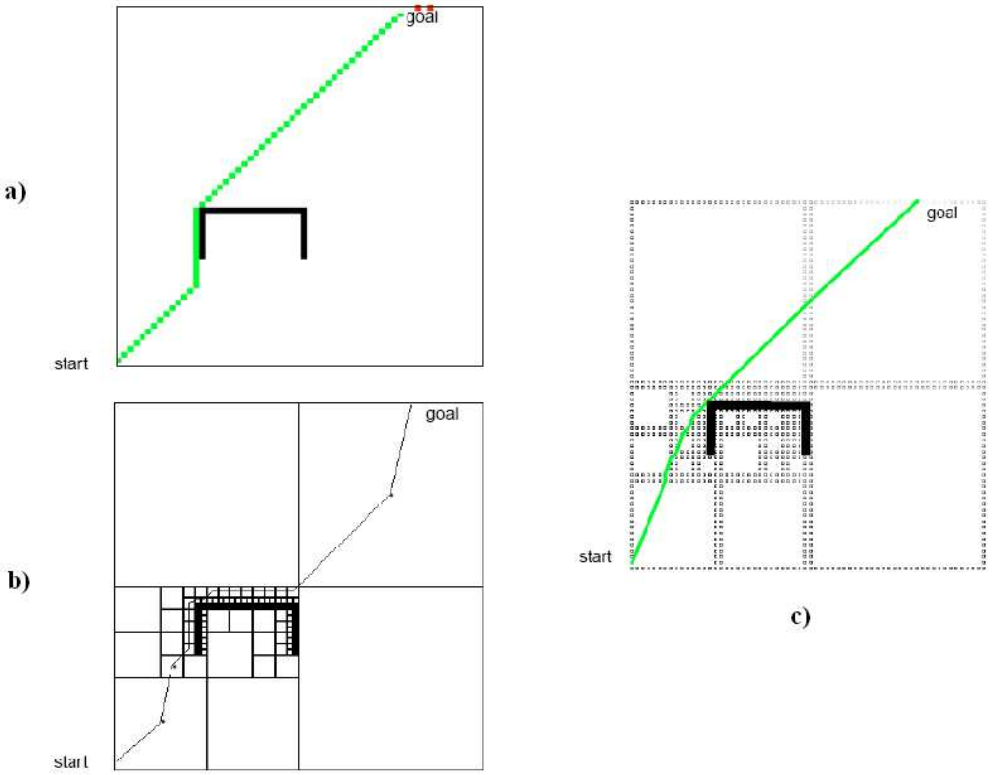

Figura 3.4: Rappresentazione con Griglie Regolari(a), con Quadtree(b) e Framed-QuadTree(c)

dati nella struttura. Generalmente, un algoritmo di questo tipo, riempie la mappa con un valore che rappresenta la probabilità di avere quella porzione occupata, incrementando questo valore ogni volta che la lettura dei sensori conferma questo evento; alcuni algoritmi decrementano il valore probabilistico nel caso in cui letture successive non confermino le precedenti. Questo tipo di approssimazioni si rende necessario a causa degli errori dei sensori di cui abbiamo parlato precedentemente.

Tutti gli algoritmi che andremo a presentare sono basati sulle griglie regolari. Inoltre, focalizzeremo la nostra attenzione, sui sensori ad ultrasuoni,

in quanto ampiamente usati nel nostro lavoro di tesi.

3.3.1 Incertezza sensoriale

I sensori usati nei robot per esplorare il mondo circostante non restituiscono con esattezza il valore che dovrebbe modellare le proprietà dell'ambiente in cui agisce. Questo è dovuto alla presenza di disturbi esterni, interferenze, oppure a causa delle condizioni strutturali e atmosferiche dell'ambiente. Un robot si trova quindi ad operare in condizioni di incertezza.

Tra i sensori più comunemente usati in questo ambito troviamo i sonar, i quali risultano essere soggetti ad errore a seconda delle condizioni atmosferiche di pressione, temperatura e umidità e si trovano in difficoltà in presenza di spigoli od oggetti composti da materiale che assorbono le onde sonore. Infatti, misurando il tempo che intercorre da quando viene emessa l'onda a quando viene ricevuta, per sapere la distanza del primo oggetto incontrato, si suppone che la velocità sia costante; ma la velocità di attraversamento dell'aria da parte del suono dipende appunto da pressione, temperatura e umidità. In più, se si incontra un oggetto composto da materiale parzialmente fonoassorbente, come il vetro per esempio, il sensore rileva una distanza maggiore di quella reale. Infine, nel caso di spigoli, l'onda sonora può rimbalzare in un' altra direzione e non giungere al sonar che identifica questa condizione come distanza infinita (si veda a tal proposito Cap. 5.2.2). [BEF96]

Altri sensori molto importanti sono quelli usati per la propriocezione (odometria), i quali restituiscono delle misurazioni che si riferiscono allo stato interno del robot. In biologia, per fare un paragone, tali valori potrebbero quantificare l'estensione di un braccio o di una gamba, in relazione ad un movimento. In robotica, molti motori hanno degli encoder che misurano le rotazioni che compiono. Conoscendo il sistema di ingranaggi e la dimensione

delle ruote, è possibile calcolare il movimento delle ruote e, di conseguenza, stimare il tragitto percorso.

Anche questi sensori, ovviamente, sono affetti da incertezza. Fattori esterni, come la struttura della pavimentazione, influenzano le misurazioni.

Per risolvere questi problemi, gli algoritmi devono usare delle approssimazioni e considerare dei valori intermedi, così da attenuare questi errori di misurazione. Altri algoritmi ancora usano dei valori di confidenza al fine di risolvere questo problema. Un valore di confidenza altro non è che la quantità di volte che una misura è stata rilevata in maniera esatta rispetto al totale: se questo rapporto è soddisfacente, la misura viene presa per buona.

3.3.2 Modello dei Sonar

Modelli per i sensori possono essere generati con metodi molto diversi:

- metodi empirici: si servono di numerose letture per provare la correttezza dei risultati. La frequenza di letture corrette porta a un valore di fiducia nell'osservazione; l'insieme dei valori di tutte le possibili osservazioni forma il modello;
- metodi analitici: generano il modello direttamente da proprietà fisiche conosciute del dispositivo;
- metodi soggettivi: si basano sull'esperienza del progettista, la quale, spesso, è un'espressione incosciente dei test empirici

Uno dei sensori che più comunemente si trova nei robot usati in questo tipo di studi, è il sonar. Il modello base di un singolo sonar ha un campo di vista, specificato nella figura 3.5, da β (metà dell'angolo che rappresenta la larghezza del cono) e R (la distanza massima che si può controllare). Questo

campo di vista può essere proiettato su una griglia regolare come evidenziato in figura 3.5. Il campo di vista può essere diviso in 3 regioni:

- regione I; dove gli elementi hanno probabilità di essere occupati
- regione II; dove gli elementi hanno probabilità di essere vuoti
- regione III; dove non si è a conoscenza della condizione di questi elementi in quanto troppo distanti dal sensore

Ovviamente le letture sono molto più corrette per quanto riguarda le celle che cadono sull'asse acustico rispetto a quelle vicino agli angoli del cono.

Mentre il modello dei sensori in Fig 3.5 è generalmente accettato, esistono molte divergenze su come convertire tale modello in un valore numerico di fiducia. I tre metodi seguenti utilizzano metodi leggermente diversi.

3.3.3 Approccio Bayesiano

Un metodo per convertire le letture dei sensori in dati numerici elaborabili è quello di tradurli in valori probabilistici, usando la regola di Bayes. Il modello dei sensori genera probabilità condizionali del tipo $P(s|H)$. Queste vengono convertite a $P(H|s)$ usando appunto la regola di Bayes. [Mur00]

Probabilità condizionali

Utilizzando H come ipotesi dell'evento, se un elemento della griglia di occupazione è occupato o vuoto ($H = Occupato, Vuoto$) possiamo ottenere $0 \leq P(H) \leq 1$.

Questa proprietà, detta non condizionata, non è molto utile per i nostri scopi, poiché non coinvolge la lettura dei sensori S. E' importante, invece, la

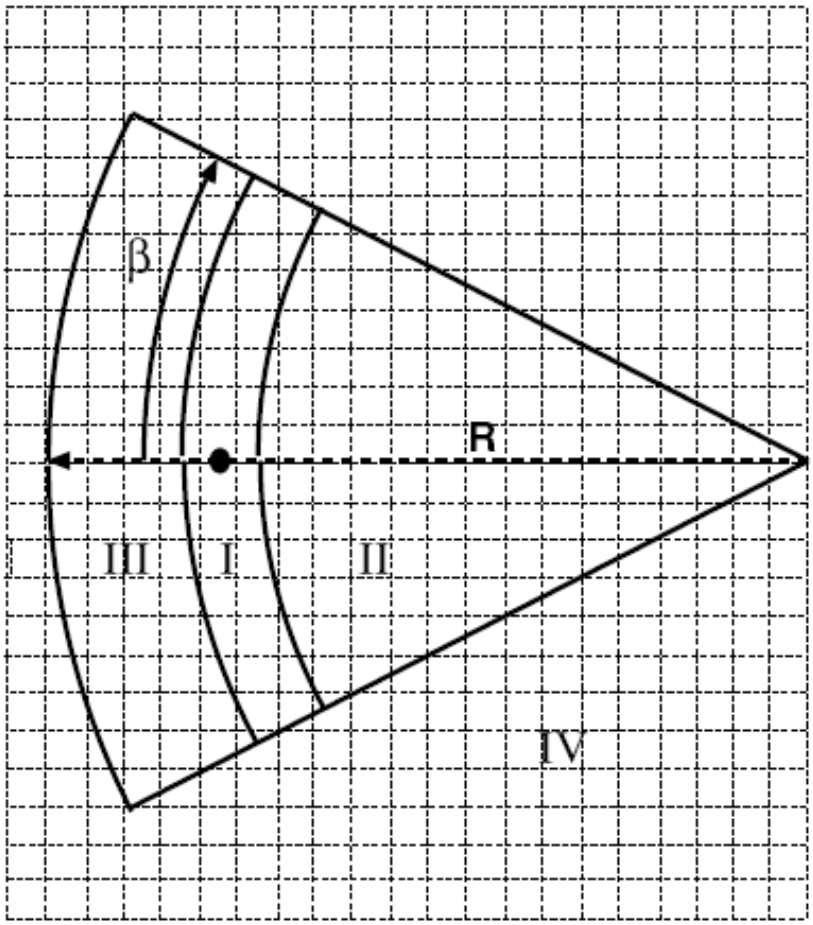

Figura 3.5: Modello dei sensori per un sonar

probabilità condizionata $P(H|s)$, che rappresenta la probabilità che la cella sia occupata in base alla lettura dei sensori effettuata.

In una griglia regolare si devono calcolare i valori $P(Occupata|s)$ e $P(Vuota|s)$ per ogni cella e i due valori vengono salvati nelle celle stesse. Questi valori rappresenteranno il valore di certezza su di una regione della griglia.

Formule di Bayes

Servono un insieme di funzioni per trasformare le letture dei sensori in probabilità:

- Per ogni elemento che cade nella regione I

$$P(Occ) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2} \times \mathrm{Max}_{Occ}$$

  e

$$P(Vuoto) = 1 - P(Occ)$$

  dove r e α sono rispettivamente la distanza e l'angolo dall'elemento della griglia; $(\beta-\alpha)/B$ cattura l'idea che gli elementi sull'asse acustico abbiano la probabilità più alta, mente $Max(Occupato)$ rappresenta la probabilità più alta che l'elemento della griglia può avere ed assume un valore compreso tra 0 e 1.

- Per ogni elemento che cade nella regione II

$$P(Occupato) = 1 - P(Vuoto)$$

  e

$$P(Vuoto) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2}$$

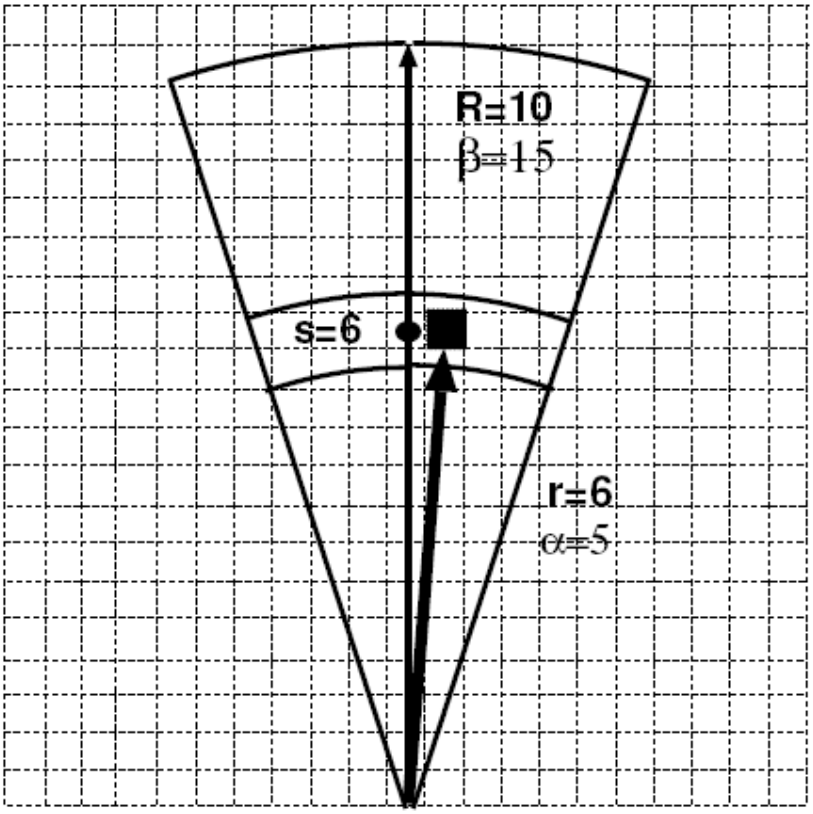

Figura 3.6: Aggiornamento di un elemento nella regione I

Per ottenere la probabilità condizionata $P(H|s)$ dobbiamo usare la regola di Bayes:

$$P(H|s) = \frac{P(s|H)P(H)}{P(s|H)P(H) + P(s|\neg H)P(\neg H)}$$

nel nostro caso diventa

$$P(Occ|s) = \frac{P(s|Occ)\mathbf{P(Occ)}}{P(s|Occ)\mathbf{P(Occ)} + P(s|Vuoto)\mathbf{P(Vuoto)}}$$

dove $P(s|Occ)$ e $P(s|Vuoto)$ sono conosciuti dal modello dei sensori.

Per quanto riguarda l'aggiornamento delle probabilità per cui la cella risulti occupata, possiamo usare la seguente versione della regola di Bayes

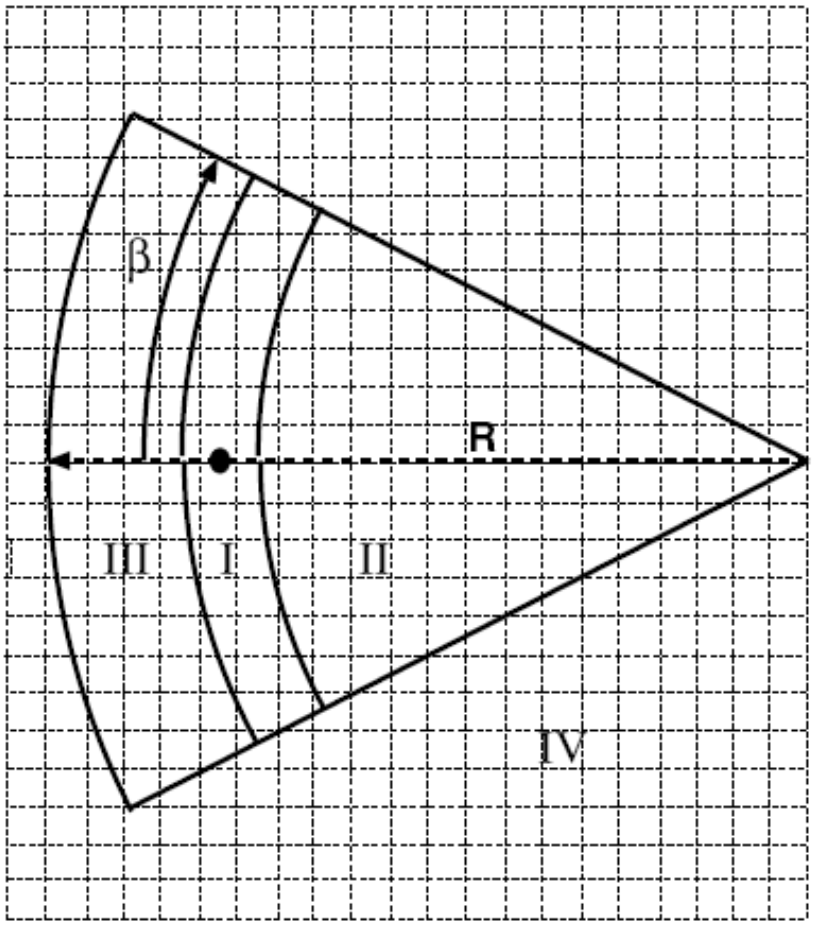

Figura 3.7: Aggiornamento di un elemento nella regione II

ricorsiva:

$$P(Occ|s_n) = \frac{P(s_n|Occ)P(Occ|s_{n-1})}{P(s_n|Occ)P(Occ|s_{n-1}) + P(s_n|Vuoto)P(Vuoto|s_{n-1})}$$

3.3.4 Teoria di Dempster-Shafer

Questo metodo produce risultati simili all'approccio Bayesiano. Usa una funzione di credibilità (detta appunto funzione di credibilità, o di *belief*, di Shafer) in aggiunta alla regola di combinazione di Dempster, molto diversa dalle funzioni di probabilità [HM92]. La regola di Dempster ha un termine

che indica quando molteplici osservazioni divergono. Questo conflitto metrico può essere usato dal robot per individuare quando la griglia di occupazione può essere soggetta ad errori.

Funzioni di credibilità di Shafer

Invece di misurare la probabilità che si verifichi una proposizione come nell'approccio Bayesiano, le funzioni di credibilità misurano la massa di belief m. Ogni sensore contribuisce con $m = 1$, ma può distribuire questa massa in svariate combinazioni di proposizioni.

Infatti, una funzione di credibilità chiama un insieme di proposizioni la "frame of discernment" (FOD), che rappresenta tutto quello che può essere acquisito da un'osservazione o sensore. Il FOD per una griglia regolare è

$$\Theta = \{Occupato, Vuoto\}$$

Il numero di tutti i possibili sottoinsiemi di FOD con cui può essere distribuita la massa di credibilità è pari a 2 elevato al numero di elementi di Θ. Nel caso di una griglia occupazionale sono {Occupato}, {Vuoto}, {Occupato, Vuoto} e empty ($\emptyset$). Il caso in cui la credibilità in una cella è {Occupato, Vuoto} significa che la cella è Occupata o Vuota; l'insieme di Θ interpreta così il "non so" associato alla lettura dei sensori.

Una funzione di credibilità (Bel) deve soddisfare le seguenti condizioni:

- $Bel(\emptyset) = 0$. Così proibisce di assegnare l'insieme vuoto, ci troviamo solo con i valori di Occupato, Vuoto o Θ.
- $Bel(\Theta) = 1$. Questo significa che $Bel(H) + Bel(\neg H) + Bel(\Theta) = 1$.
- Per ogni intero positivo n e ogni collezione $A1, \ldots, An$ dei sottoinsiemi

di Θ,

$$Bel(A1, \ldots, An) \geq \sum_{\mathcal{I} \subset \{1,\ldots,n\}; \mathcal{I} \neq \emptyset} (-1)^{|\mathcal{I}|+1} Bel\left(\bigcap_{i \in \mathcal{I}} A_i\right)$$

Questo significa che le funzioni di credibilità che contribuiscono per Θ possono essere sommate e che il risultato della credibilità in una proposizione può essere più alto dopo la somma.

Ogni elemento della griglia ha quindi 3 valori di belief:

$$Bel = m(Occupato), m(Vuoto), m(Incerto).$$

Queste sono le formule relative ai sonar:

- Per ogni elemento che cade nella regione I

$$m(Occ) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2} \times \mathrm{Max}_{Occ}$$

$$m(Vuoto) = 0$$

$$m(Incerto = 1 - m(Occ)$$

- Per ogni elemento che cade nella regione II

$$m(Vuoto) = \frac{\left(\frac{R-r}{R}\right) + \left(\frac{\beta-\alpha}{\beta}\right)}{2}$$

$$m(Occ) = 0$$

$$m(Incerto) = 1 - m(Vuoto)$$

La regola di combinazione di Dempster

Nella teoria ci sono molti modi per combinare le funzioni di credibilità. La più popolare è la regola di combinazione di Dempster. Essa tratta la combinazione di 2 funzioni di credibilità come una intersezione fisica. Questo significa che la regola può essere applicata a osservazioni sovrapposte nel tempo. Due funzioni di credibilità possono essere rappresentate ognuna come un segmento di lunghezza 1 corrispondente ad una quantità di massa di belief. Il segmento può essere diviso nella massa associata con ogni elemento di Θ. Nella regola di Dempster, i 2 segmenti formano assi ortogonali, creando un quadrato di area 1. L'interno del quadrato può essere diviso in sotto-regioni rappresentanti la belief nell'elemento di Θ prodotto dall'insieme di intersezioni tra i 2 assi (vedere fig. 3.8).

Le sotto-regioni di un quadrato di area 1 possono essere proiettate in un segmento di lunghezza 1. Questo significa che l'area di massa di belief creata dall'intersezione di 2 funzioni forma una terza funzione di credibilità. La massa di belief per *Occupato* è presa dalla somma delle aree di ognuna delle sotto-regioni che può essere occupata.

Siccome ci sono proposizioni mutualmente esclusive, come si vede in fig. 3.8, alcuni insiemi possono risultare vuoti; questo è un problema perché va contro ad una delle regole delle funzioni di belief di Shafer. La regola di Dempster risolve il problema normalizzando la funzione di credibilità: l'area vuota viene distribuita equamente ad ogni area non vuota.

Queste sono le formule per ricavare i valori delle masse di belief:

$$m(Occ) = \frac{\sum_{A_i \cap B_j = Occ} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)}$$

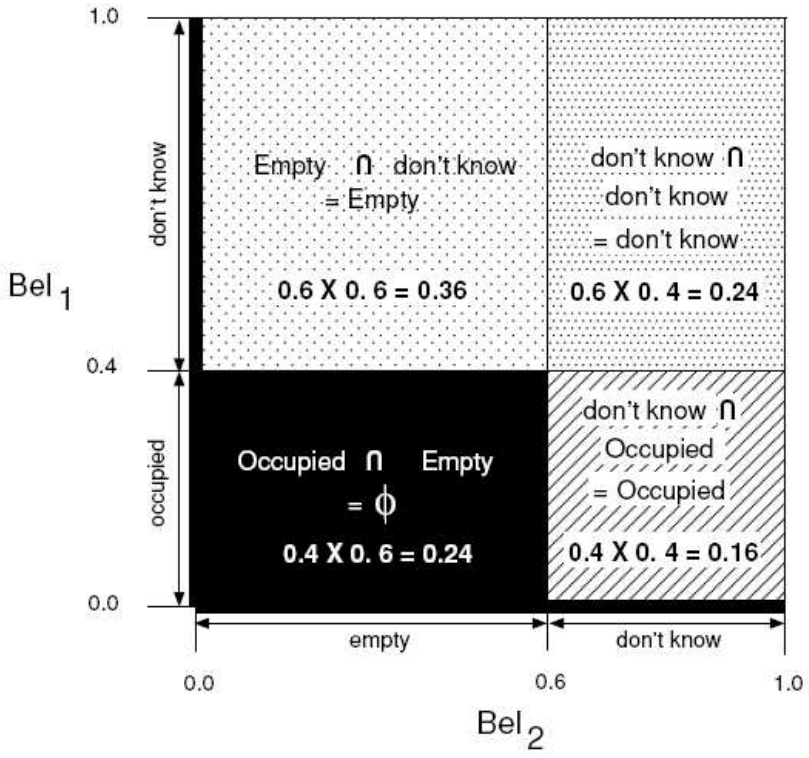

Figura 3.8: Massa di belief associata

$$m(Vuoto) = \frac{\sum_{A_i \cap B_j = Vuoto} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)}$$

$$m(Incerto) = \frac{\sum_{A_i \cap B_j = Incerto} m(A_i)m(B_j)}{1 - \sum_{A_i \cap B_j = \emptyset} m(A_i)m(B_j)}$$

3.3.5 HIMM

L'algoritmo Histogrammic in Motion Mapping (HIMM) assegna punteggi di Occupato-Vuoto alle celle di una griglia regolare in modo diverso dagli altri

metodi. L'HIMM è stato sviluppato per avere un algoritmo veloce e leggero da applicare in sistemi di "obstacle avoidance". Questo algoritmo, infatti, permette di costruire una mappa pur facendo spostare velocemente il robot, anzi, come vedremo fra breve, ottiene risultati migliori quanto più la velocità è sostenuta.[BK91]

Modello dei sonar e regola di aggiornamento

L'HIMM utilizza un semplice modello per i sonar, come si evince dalla fig. 3.9. In questo modello si possono aggiornare solo le celle che si trovano lungo l'asse acustico. Questo elimina più del 90% degli elementi della griglia da aggiornare rispetto agli altri due metodi precedenti, riducendo drasticamente l'ordine di complessità dell'algoritmo. E' importante notare come le letture dai sonar avvengano nella stessa maniera sia nel metodo Bayesiano, sia in quello di Dempster-Shafer e sia nell'HIMM; quello che cambia è che quest'ultimo interpreta le informazioni dai sonar in maniera differente e su molti meno elementi rispetto agli altri due metodi. Un'altra differenza è che il punteggio di incertezza è espresso tramite un intero compreso tra 0 e 15 e questo significa che ogni elemento può essere rappresentato da un solo byte, invece che da una struttura che deve contenere almeno due numeri in virgola mobile.

Il modello dei sonar rappresentato in 3.9, mostra anche le regole di aggiornamento. Nell'HIMM, il punteggio viene incrementato di un valore I ogni volta che l'asse acustico di lettura del sonar passa su di una cella. Se tale cella è vuota l'incremento I vale -1, altrimenti I è $+3$. In più, la regola di aggiornamento, come già detto, è computazionalmente efficiente, visto che è composta o da una sottrazione o da una somma di interi.

La formula base è la seguente:

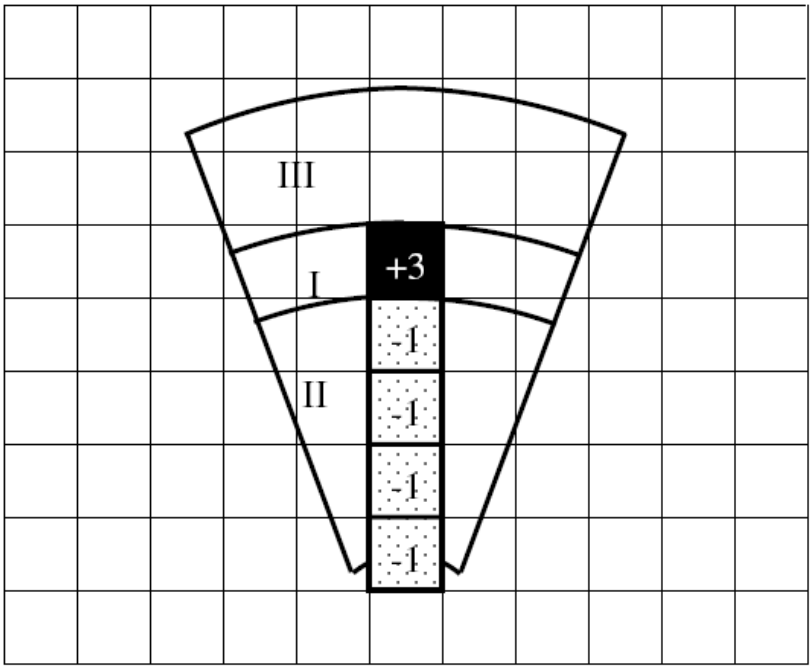

Figura 3.9: Modello dei sonar dell'HIMM

$$grid[i][j] = grid[i][j] + I \quad \text{dove } 0 \leq grid[i][j] \leq 15$$

Se risulta chiaro come mai l'HIMM è molto più veloce dei precedenti metodi, non lo è altrettanto per quanto riguarda il produrre griglie per la navigazione più affidabili. L'aggiornamento lungo il solo asse acustico produce un campione di risultati abbastanza piccolo, come mostrato dalla figura 3.10. Viene mostrato un muro (in grigio), che dovrebbe apparire sulla griglia come una serie di piccoli isolati "paletti". C'è il grave rischio che il robot possa considerare valido un percorso che attraversa questi paletti, quando nella realtà è presente un muro. Se il robot è in movimento e la frequenza di aggiornamento della griglia è in accordo con la sua velocità, tali lacune vengono colmate. L'HIMM lavora meglio quando il robot si muove a grande velocità. Se la velocità e l'aggiornamento della griglia sono ben calibrati, la

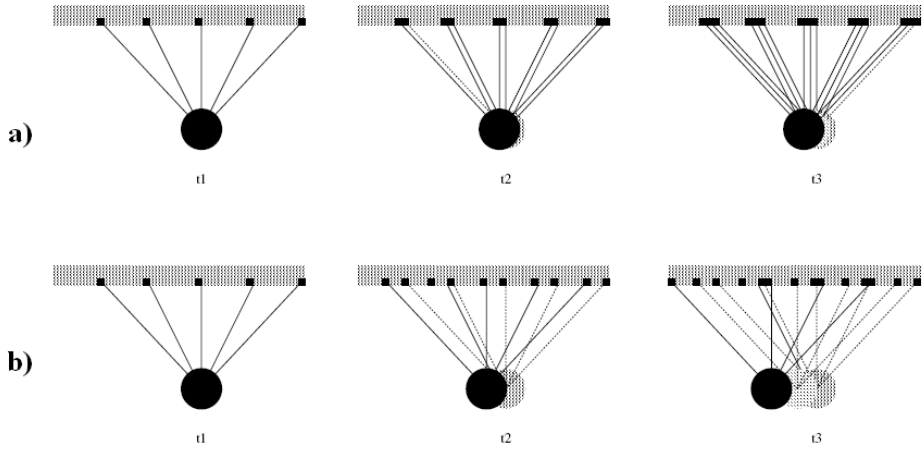

Figura 3.10: Esempio di aggiornamento di un muro con HIMM. a) velocità e frequenza di aggiornamenti sono ben bilanciati. b) la frequenza di aggiornamento è più lenta della velocità del robot, produzione di buchi

distanza tra i punti del muro sarà minimizzata e il problema risolto (vedere figura 3.10a). In caso contrario, appariranno dei buchi (figura 3.10b), in quanto il robot percepirà troppo poco del mondo che lo circonda.

Growth Rate Operator

Uno svantaggio dell'HIMM è che si possono ottenere delle celle occupate, ma sparse e isolate lungo tutta la griglia, questo avviene perché un elemento potrebbe subire solo uno o due aggiornamenti. Si hanno pertanto due conseguenze. La prima è che i valori nella griglia tendono ad essere bassi; un elemento che è percepito come occupato, ottiene solo due aggiornamenti e avrà un punteggio di 6, che è meno della metà del punteggio massimo di 15. Secondo, piccoli ostacoli come paletti o sedie, che mostrano un piccolo profilo ai sonar, non riceveranno mai un punteggio alto. Una soluzione potrebbe

essere quella di aumentare il valore degli incrementi, in questo modo, però, il processo di aggiornamento tenderà a produrre mappe con molti oggetti fantasma.

Un altro approccio è quello di considerare i valori delle celle vicine e quindi, di usare un'euristica secondo la quale se molti dei vicini di un elemento risultano occupati, molto probabilmente lo sarà pure lui e di conseguenza, il suo punteggio dovrà essere incrementato. Si noti che lo stesso risultato veniva ottenuto nei metodi precedenti, adottando, nel modello dei sonar, un ampio valore di β.

Il meccanismo che utilizza l'HIMM, invece, è chiamato Growth Rate Operator (GRO): quando un elemento $grid[i][j]$ è aggiornato con una lettura di occupato, il GRO viene applicato come passo extra. Il GRO usa una maschera W definita dalle celle vicine. La maschera non è altro che una matrice quadrata di valori che viene sovrapposta alla griglia di occupazione. Il centro della maschera si posiziona su $grid[i][j]$. Ad esempio, si può usare una matrice 3x3 che tiene in considerazione gli 8 vicini intorno all'elemento. Più grande è la maschera, più rapida sarà la crescita del valore della cella e l'influenza dei vicini. In più, la crescita dell'elemento avrà a sua volta influenza sui vicini.

La maschera contiene un fattore di peso per ogni cella. Per esempio, si può pensare di avere una maschera 3x3 con valore 1 al centro e 0.5 per i vicini.

A questo punto la cella viene aggiornata in 2 passi; nel primo si avrà

$$grid[i][j] = grid[i][j] + I$$

questo è l'aggiornamento standard dell'HIMM che produce un valore tempo-

raneo che sarà usato nel passo successivo,

$$grid[i][j] = \sum_{p,q=-1...1} grid[i][j] \times W[i+p][j+q]$$

questo aggiornamento somma i valori delle celle dell'intorno, moltiplicate per il loro peso nella maschera.

Nella figura 3.11 si vede un esempio di computazione dell'HIMM con il GRO.

Il vantaggio del GRO è quello di essere molto sensibile agli ostacoli potenziali. Ma questo ha anche degli svantaggi: gli ostacoli tendono ad essere più grandi rispetto alla realtà. Ciò avviene perché gli errori dei sonar aggiunti a quelli odometrici comportano delle letture molto ampie attorno ai piccoli oggetti. Quando gli errori nelle letture sono minimi, il GRO innalza i valori delle celle già aggiornate dalla presenza dell'oggetto percepito. Questo problema è comunque irrilevante se il robot non si muove in aree anguste e strette.

3.4 Path Planning

Supponendo di aver scelto le griglie regolari come struttura dati per la mappa e di averla costruita (magari con l'HIMM che è diventato uno standard de facto), si deve trovare un metodo per pianificare il percorso che il robot dovrà seguire per giungere al suo obiettivo.

Per far questo, conviene convertire la griglia ottenuta in un grafo, avvalendoci in tal modo della teoria sui grafi, possiamo applicare algoritmi di ricerca di cammini minimi. Infatti, trasformare la griglia in un grafo, significa poter usare algoritmi di ricerca su grafi per calcolare un cammino tra il nodo iniziale ed il nodo obiettivo. Molti di questi algoritmi richiedono la visita di

ogni nodo del grafo per determinare il percorso più breve. Visitare ogni nodo può essere computazionalmente trattabile per grafi poco connessi come quelli derivati da grafi di Voronoi, ma diventa troppo costoso computazionalmente per grafi altamente connessi come quelli derivati da griglie regolari.[Ste99]

C'è una classe di algoritmi che ha applicato l'idea di tagliare i percorsi che non sono ottimali. Tra questi algoritmi c'è l'A* (anche noto come algoritmo di Dijkstra).

3.4.1 Algoritmo A*

Supponiamo di assegnare a ciascuno degli archi a di un grafo orientato G, un certo peso intero e positivo p_a. In questo modo, ai cammini nel grafo, ossia a sequenze di archi consecutivi, verrà assegnato un peso dato dalla somma dei pesi degli archi che lo compongono. Dati due nodi x e y, il problema del cammino minimo consiste nel fornire un cammino da x a y di peso minimo.

Edsger W. Dijkstra ha proposto nel 1959 [Dij59] un algoritmo che risolve questo problema. L'algoritmo di Dijkstra visita i nodi nel grafo, in maniera simile a una ricerca in ampiezza o in profondità. In ogni istante, l'insieme $\mathcal{N}$ dei nodi del grafo è diviso in tre parti: l'insieme dei nodi visitati $\mathcal{V}$, l'insieme dei nodi di frontiera $\mathcal{F}$, che sono nodi raggiungibili lungo un arco uscente dai nodi visitati (nodi successori di quelli visitati), e i nodi sconosciuti, che sono ancora da esaminare. Per ogni nodo z, l'algoritmo tiene traccia di un valore d_z, inizialmente posto uguale a 1, e di un nodo u_z , inizialmente indefinito. L'algoritmo consiste semplicemente nel ripetere il seguente passo: si prende dall'insieme $\mathcal{F}$ un qualunque nodo z con d_z minimo, si sposta z da $\mathcal{F}$ in $\mathcal{V}$, si spostano tutti i successori di z sconosciuti in $\mathcal{F}$, e per ogni successore w di z si aggiornano i valori d_w e u_w. L'aggiornamento viene effettuato con la

regola

$$d_w = min\{d_w, d_z + p_a\}$$

dove "a" è l'arco che collega z a w. Se il valore di d_w è stato effettivamente modificato, allora u_w viene posto uguale z.[Vig06]

La regola segue un'idea piuttosto naturale: se sappiamo che con peso d_z possiamo arrivare fino a z, allora arrivare a w non può costare più di arrivare a z e spostarsi lungo un arco fino a w.

L'aggiornamento di u_w ci permette di ricordare che, al momento, il cammino di peso minimo che conosciamo per arrivare da x in w ha come penultimo nodo z.

L'algoritmo inizia con $\mathcal{V} = \emptyset$, $\mathcal{F} = x$ e $d_x = 0$ e prosegue finché y non viene visitato, o finché $\mathcal{F} = \emptyset$: in questo caso, y non è raggiungibile da x lungo un arco orientato. Se usciamo solo nel primo caso, alla fine dell'algoritmo d_z contiene, per ogni nodo z, il peso di un cammino minimo da x a z; inoltre, il vettore u permette di ricostruire l'albero dei cammini minimi con origine in x.

Quando il robot si muove sulla mappa, confronta gli ostacoli contenuti in essa con i valori che vengono percepiti dai propri sensori. Una volta che riscontra un ostacolo non modellato, può aggiornare la mappa e di conseguenza anche il grafo; una volta modificato il grafo bisogna ricostruire il percorso. Un altro modo per affrontare il problema è l'algoritmo D*.

3.4.2 Algoritmo D*

L'algoritmo D* è stato creato appositamente per operare in ambienti dinamici ed è funzionalmente equivalente a ripianificatori del percorso ottimale classici, ma molto più efficiente. [Ste95]

L'idea del D* è quella di eseguire l'algoritmo A* da ogni possibile locazione fino al nodo obiettivo. Questo è computazionalmente costoso, ma essendo eseguito una volta sola all'avvio, non è un gran problema.

Così facendo, questo algoritmo aiuta un comportamento di tipo reattivo perché, una volta localizzata la posizione del robot sulla mappa, quest'ultimo può prendere il percorso più breve per l'obiettivo (che è già stato computato). Questo permette ad un robot che si è spostato ad un sotto-obiettivo per evitare un ostacolo non modellato, di prendere il percorso più breve per raggiungere l'obiettivo dalla nuova posizione senza dover ripianificare il percorso ottimale.

Nel caso in cui ci siano molti ostacoli non modellati, il robot sarebbe costretto a cambiare spesso sotto-obiettivo per evitare i vari ostacoli, magari senza fare reali progressi. La soluzione del D* è quella di aggiornare continuamente la mappa e dinamicamente riaggiornare i vari percorsi con A* (ripianificazione continua). Questo ha 2 svantaggi:

- è computazionalmente costoso;
- è altamente dipendente dalla qualità dei sensori.

Esistono delle estensioni del D* che riducono il numero di percorsi da calcolare ad ogni ripianificazione, aggiornando solo quelli che possono essere utili per arrivare all'obiettivo dalla posizione attuale. Il termine "essere utili" nasconde ovviamente una euristica che l'algoritmo segue nella scelta. Questa estensione del D* (Focussed D*) rende l'algoritmo più leggero.

3.5 Localizzazione

Per navigare in maniera affidabile, un robot deve necessariamente sapere dove si trova. Per raggiungere questo obiettivo sono stati sviluppati diversi

algoritmi. [Mur00]

Gli algoritmi di localizzazione si possono dividere tra quelli che usano la propriocezione, cioè, che per stabilire la posizione attuale, si affidano ai dati degli encoder interni abbinati ad informazioni sullo stato, come posizione precedente e movimenti effettuati, e quelli che usano l'esterocezione, ossia si basano sull'osservazione dell'ambiente. Negli algoritmi che usano esterocezione, la localizzazione può, sia usare direttamente i dati dei sensori (iconic), sia usare caratteristiche estratte da questi dati (feature-based). Per la localizzazione iconic, il robot si muove per una piccola distanza e poi confronta quello che "vede" con quello che c'è sulla mappa. Il confronto con la mappa è ulteriormente complicato dall'incertezza nella mappa di occupazione in quanto costruita su dati, spesso non esatti, dei sensori. La localizzazione feature-based si basa sulla distinzione di oggetti o parti dell'ambiente che può facilmente riconoscere dai dati dei sensori e da quelli, sapendo come sono disposti sulla mappa, sapere dove si trova. Un problema che si può avere è che il robot può confondere una parte dell'ambiente per un'altra molto simile, come ad esempio due diverse porte.

Il vantaggio della localizzazione iconic è quella di essere più accurata e che impone meno restrizioni sull'ambiente (come la conoscenza a priori delle caratteristiche dello stesso); gli svantaggi sono quelli di essere computazionalmente più costosa e più lenta e, nel caso di poche informazioni sulla locazione iniziale, anche meno precisa.

Comunque sia, la localizzazione in un ambiente molto dinamico è abbastanza difficile per qualsiasi algoritmo perché rende impossibile il confronto tra la passata e la corrente osservazione.

3.5.1 Localizzazione continua e costruzione della mappa

Una volta che il robot conosce la sua posizione rispetto alla mappa, la percezione corrente viene aggiunta alla mappa in un processo chiamato registrazione.

Quando il robot si muove da una posizione ad un'altra, si può basare sui suoi encoders interni per sapere dove si trova. Siccome, però, gli encoders generano errori su tutti e tre i gradi di libertà, ossia sugli assi cartesiani, la reale posizione del robot può essere diversa da quella rilevata. Questo porterebbe ad un incremento del numero di possibili confronti che il robot deve fare. Gli encoders, invece, provvedono a presentare un insieme di possibili locazioni e orientazioni; ogni possibile orientazione e posizione è chiamata posa.

La costruzione di una mappa locale permetterebbe una buona localizzazione nel caso di movimenti brevi, soprattutto se successivamente i risultati si confrontassero su una mappa globale, ma rimane comunque il problema degli errori nella lettura dei sensori che si ripercuotono su tutta la procedura di confronto. Una frequente localizzazione è da preferirsi, perché, più breve è il movimento fatto dal robot, minore è il numero di pose da considerare e maggiori sono le intersezioni tra la passata e la corrente percezione. Per evitare errori di rumore tra i sonar o si aggiorna la lettura per ogni sensore, o lo si fa per un sensore alla volta. Da dati sperimentali, si è visto che è migliore la seconda possibilità.

Come detto, per migliorare la localizzazione si può costruire una mappa locale; questo lo si può fare sfruttando le passate letture dall'ultimo aggiornamento. Dopo n letture, la griglia locale viene confrontata con quella globale: un confronto per ogni possibile posa. Il numero di pose da considerare deve tener conto del range del possibile errore commesso dai sensori e dalla discre-

tizzazione dello stesso per ridurre il numero di pose che sarebbe, altrimenti, infinito.

Il confronto avviene posizionando il centro della griglia locale sulla cella della griglia globale individuata da ogni possibile posa. Questa operazione permette di marcare un insieme di elementi della griglia che si sovrappongono in entrambe le griglie. A questo punto, l'algoritmo assegna un punteggio ad ogni confronto in base a queste sovrapposizioni che sarà il valore di incertezza. Una volta che sono stati eseguiti e valutati tutti i confronti, l'algoritmo sceglie la posa con il miglior punteggio. Questa posa viene presa come posizione corrente del robot sulla griglia globale e usata per aggiornare l'odometria dello stesso.

3.5.2 Localizzazione Monte Carlo

La localizzazione Monte Carlo (MC) è un approccio relativamente nuovo al problema della localizzazione. L'MC si basa su una collezione di pose. Ogni posa consiste di una possibile locazione che il robot può correntemente occupare e un valore che rappresenta la probabilità (o peso) che il robot sia effettivamente in questa locazione.[DFBT99]

Più pose si hanno e più è veloce l'algoritmo a raggiungere la soluzione corretta, ma il costo computazionale aumenta, e diminuisce la velocità del robot.

Quando l'algoritmo inizia, il robot non sa dove si trova, così la posa corrente è egualmente distribuita su tutto l'arco delle possibili locazioni, e lo stesso vale per i pesi loro associati. Con l'avanzare del tempo, le pose vicine alla posizione corrente dovrebbero diventare più probabili. Se usiamo una linea grafica, la quale disegna le pose usando il peso delle possibili locazioni, il grafico dovrebbe evidenziare la posizione attuale. Pensando questa curva

come rappresentazione dello stato delle possibilità sulla locazione del robot, all'inizio avremo una linea piatta, ma dopo un po' di tempo, mostrerebbe delle gobbe sulle locazioni più probabili, con una punta larga sulla posizione attuale.

Quindi l'idea generale è la seguente:

- Inizializzare l'insieme di pose (le pose correnti) così che le loro locazioni siano egualmente distribuite e i loro pesi siano uguali.
- Ripetere fino alla fine con il corrente insieme di pose:
  - ♢ Muovere il robot di una distanza fissa e poi prendere una lettura dei sensori
  - ♢ Aggiornare la locazione di ognuna delle pose (usando il modello del movimento)
  - ♢ Assegnare i pesi di ogni posa alla probabilità che questa lettura dei sensori dà alla nuova locazione (usando il modello dei sensori).
  - ♢ Creare una nuova collezione di pose dal corrente insieme di pose basandosi sui loro pesi.
  - ♢ Passare questa nuova collezione come collezione corrente.

Bisogna ricordare che l'attuale distanza di movimento potrebbe non essere la distanza che ci si aspettava. Così, il modello deve aggiungere o sottrarre un fattore di errore casuale.

Quando si crea una nuova collezione di pose, si sceglie di togliere le locazioni con basso peso e inserirne altre con peso più alto; così facendo si ha un approccio tipico degli algoritmi genetici: sopravvive il più grande. Alla fine il robot si troverà nella locazione o area con più pose.

3.5.3 Markov Localization

La Markov Localization (ML) fornisce un framework probabilistico generale per la stima della posizione globale basato sulle osservazioni e le azioni. Si usa il termine "Markov Localization" perché è una diretta applicazione della stima dello stato all'interno del framework "Partially observable Markov decision processes" (POMDP). Questo framework ha origine nel dominio delle operazioni di ricerca dove è usato per creare decisioni ottimali quando azioni e letture dei sensori non sono ottimali. Quando si applica POMDP a un problema di pianificazione, lo stato del mondo nel quale l'agente lavora è modellato da una o più variabili casuali e le prestazioni della decisione presa dall'agente è misurata dall'utilità degli stati del mondo.[Fox98]

Una delle precondizioni principali per prendere le decisioni è la conoscenza dello stato del mondo. Sfortunatamente, in molti ambienti realistici, lo stato di interesse, non può essere osservato direttamente ma può solo essere stimato basandosi sull'incerta informazione ricavata dai sensori. Quindi, un importante componente dei POMDP è lo state estimator, che ha il compito di stimare lo stato del mondo basandosi sui dati dei sensori e sulle azioni prese dall'agente.

La ML è un caso speciale di uno state estimator: l'agente è un robot mobile e lo stato del mondo è la posizione del robot all'interno dell'ambiente. Con L_t denotiamo lo stato del robot al tempo t. Con $Bel(L_t = l)$ denotiamo la valutazione del robot se la sua localizzazione era l al tempo t. Questa valutazione rappresenta una distribuzione di probabilità sull'intero spazio di L.

Lo stato rappresentato da L_t viene aggiornato ogni volta che è ricevuto un input dai sensori o il robot si è mosso. Senza perdita di generalità, possiamo assumere che ad ogni punto discreto i nel tempo, viene percepita una misu-

razione S_i e un'azione A_i viene eseguita (l'indice i viene incrementato dopo ogni azione). Il compito di ML è l'aggiornamento della valutazione $Bel(L_t)$ per ogni locazione l dell'ambiente come il robot si muove.

In generale, lo stato al tempo t è condizionato da tutti i dati disponibili dati da $Bel(L_t|S_1, \ldots, S_t, A_1, \ldots A_{t-1})$ che rappresenta la valutazione data a tutte le azioni A_i e percezioni S_i precedenti. La computazione della probabilità condizionale cresce esponenzialmente nel tempo con il numero di condizioni variabili.

Risolvendo la probabilità condizionale con la formula di Bayes la valutazione diventa

$$Bel(L_t = l|s_{1,\ldots,t}, a_{1,\ldots,t-1}) =$$

$$= \frac{P(s_t|L_t = l, s_{1,\ldots,t-1}, a_{1,\ldots,t-1})P(L_t = l|s_{1,\ldots,t-1}, a_{1,\ldots,t-1})}{P(s_t|s_{1,\ldots,t-1}, a_{1,\ldots,t-1})}$$

Infine, l'ambiente nel quale il robot va a muoversi deve seguire le seguenti assunzioni affinché sia possibile utilizzare la ML:

- indipendenza delle azioni; la conoscenza sulla posizione al tempo $t-1$ e sul comando di movimento eseguito dal robot al tempo $t-1$ sono sufficienti per predire la posizione del robot al tempo t. Questa assunzione è ragionevole nel contesto della localizzazione di un robot mobile, perché tutte le azioni e posizioni precedenti a $t-1$ non aggiungono informazione alla posizione corrente del robot.

- indipendenza dalle percezioni; tutte le percezioni dipendono dalla posizione del robot nell'ambiente. Quindi, data la posizione del robot, la distanza misurata dai sensori di prossimità dipende solo dalla distanza

dagli ostacoli nell'ambiente e non dipendono dalla distanza misurata da altri sensori di prossimità.

3.6 Evitare gli ostacoli

Lo scopo di ogni sistema di navigazione è quello di raggiungere un luogo obiettivo partendo da una locazione di partenza nel minor tempo possibile. Nel fare questo, però, deve riuscire ad evitare gli ostacoli che incontra durante il suo cammino. Esistono quindi diversi approcci una volta che un nuovo ostacolo è stato individuato:

- il robot aggiorna la mappa, crea un nuovo percorso ottimale per raggiungere l'obiettivo e parte seguendo il nuovo path.
- il robot realizza un path per raggiungere un punto sotto-obiettivo per evitare l'ostacolo, e una volta lì, crea un path per raggiungere l'obiettivo principale.

Un accorgimento che si potrebbe adottare, sarebbe quello di considerare come celle da evitare anche le celle vicine a quelle percepite come "nuovo ostacolo", in modo tale da creare una sorta di "cuscinetto" tra il nuovo oggetto e il robot e permettendo in tal modo a quest'ultimo di aggirare l'ostacolo con un certo margine di sicurezza ed evitando così, o almeno riducendo, possibili e inefficienti cicli di operazioni per eludere sempre lo stesso ostacolo nel caso in cui questo occupi molte più celle di quelle inizialmente percepite. D'altronde, se il robot opera in spazi stretti, tipo un corridoio, ingrandire ostacoli, potrebbe portare al blocco di un intero percorso, in realtà valido, spingendo il robot alla scelta di un path alternativo molto più lungo o addirittura a stabilire l'irraggiungibilità dell'obiettivo. Si può concludere quindi

che questo accorgimento è da adottarsi senza particolari problemi in ambienti ampi, come quelli esterni, mentre in quelli interni bisognerebbe cercare altre soluzioni e magari optare per una tecnica di "avoiding" più precisa e accurata.

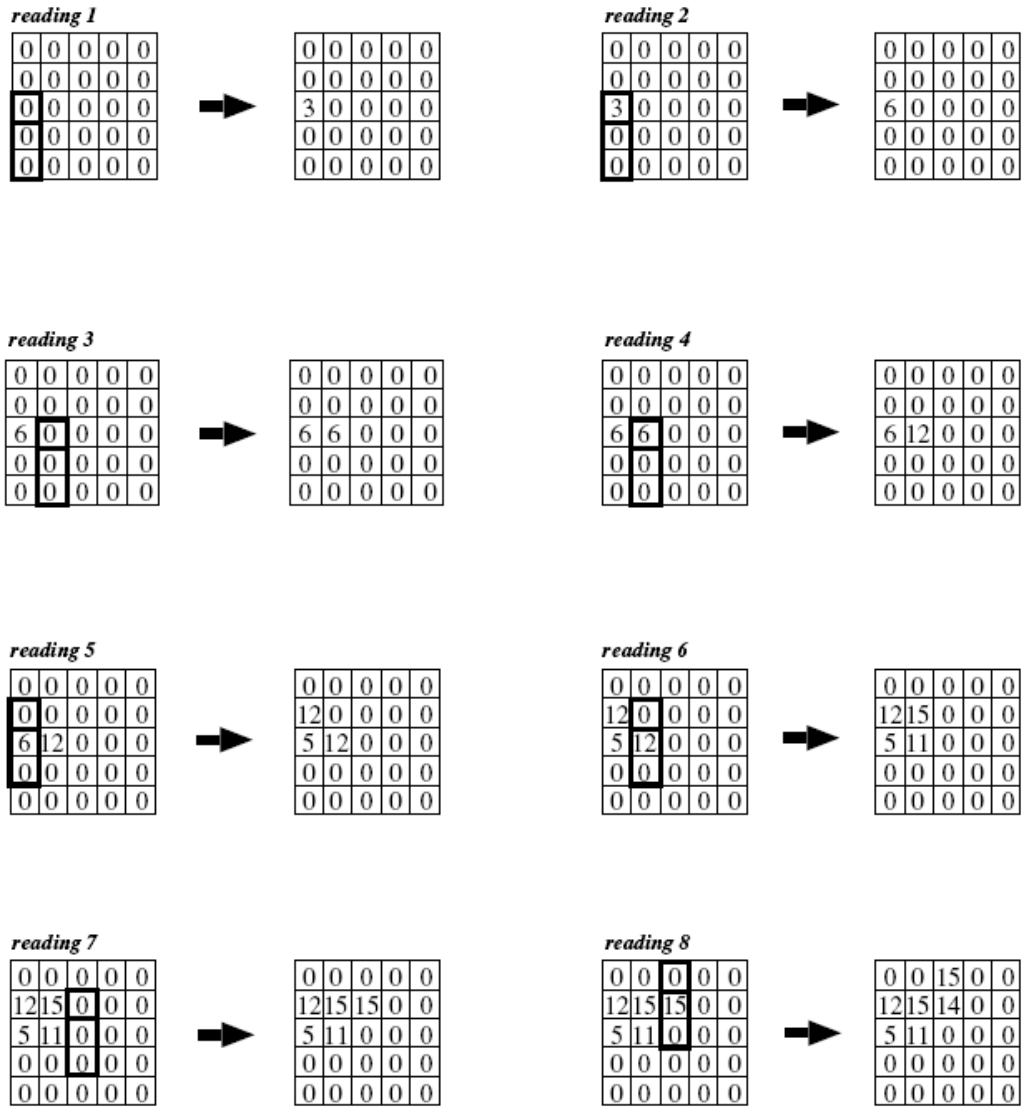

Figura 3.11: Aggiornamento con GRO

Capitolo 4

Implementazione di un sistema di navigazione basato su anticipazioni sensoriali

Uno dei principali problemi tradizionalmente affrontati nel settore della navigazione autonoma, è quello di dotare i robot della capacità di percepire l'ambiente circostante per svolgervi il proprio compito e per reagire con sufficiente tempestività alle sue variazioni.

L'applicazione del paradigma reattivo al problema della navigazione è la soluzione più frequentemente adottata, anche se spesso combinata con approcci deliberativi, in architetture ibride [Mur00].

Il paradigma puramente reattivo, infatti, può non portare a risultati soddisfacenti in termini di prestazioni. Questo è tanto più evidente quando i dati da elaborare risultano computazionalmente complessi (come ad esempio nel caso della visione artificiale).

Con l'applicazione del paradigma anticipatorio alla navigazione, si punta ad eliminare almeno in parte questo inconveniente. La possibilità di predire la

risposta sensoriale, utilizzando opportuni modelli interni, permette di evitare tutte quelle operazioni di processing necessarie a trasformare i dati grezzi in informazioni fruibili dall'algoritmo di pianificazione, almeno nel caso in cui i valori predetti e quelli effettivi risultino concordi.

Lo schema dell'Expected Perception rappresentato in figura 2.13 deve essere quindi riferito al problema della navigazione.

I sonar sono i sensori più comunemente adottati per la risoluzione di questo tipo di problema e di conseguenza, le applicazioni sviluppate nel presente lavoro di tesi fanno riferimento ad essi. I dati che si dovranno manipolare saranno tipicamente dei valori in virgola mobile, relativi alle letture di ogni singolo sonar e che si riferiscono alla distanza di un eventuale ostacolo percepito.

Il modello interno, essenziale al paradigma anticipatorio, non può che essere la rappresentazione del mondo percepito dal sonar. Gli ostacoli quindi sono definiti come presente/assente. Considerando questo aspetto, risulta conveniente esprimere la mappa attraverso una Griglia Regolare.

Il modulo di predizione, utilizzando sia i dati relativi al robot, quali posizione e velocità, sia i dati della mappa, dovrà estrapolare i valori numerici che i sonar daranno nell'istante successivo. Il confronto tra la percezione attesa e quella reale è semplicemente una differenza tra questi valori numerici.

Rispetto all'adozione di un paradigma "puramente" reattivo, si potrà avere quindi un risparmio computazionale pari alla procedura di analisi dei dati sensoriali, perlomeno nei confronti degli ostacoli già presenti nel mondo.

Per lo sviluppo del sistema di navigazione nel suo complesso, sono state create due applicazioni. La prima, chiamata *MapBuilding*, dedicata alla creazione della mappa, che costituirà il modello interno dell'Expected Perception. La seconda, *Navigation*, è l'applicazione di navigazione vera e propria.

Il sistema di navigazione è stato realizzato su piattaforma Java 1.6. Per il salvataggio della mappa dell'ambiente è stato utilizzato il database Postgress 8.2.1. Le due applicazioni MapBuilding e Navigation hanno una struttura generale simile, costituita da un modulo di elaborazione, distinto per le due applicazioni, un modulo di lettura e uno di scrittura, che, rispettivamente, ricevono ed inviano comandi al robot. Questi due moduli costituiscono a tutti gli effetti un'interfaccia verso il robot, in questa maniera il modulo di elaborazione può essere implementato indipendentemente dal tipo di robot utilizzato. L'architettura generale verrà approfondita ulteriormente nel paragrafo 4.4.

Nei paragrafi successivi, verrà esposto nel dettaglio il funzionamento e le scelte implementative di ogni applicazione.

4.1 MapBuilding

Come spiegato in precedenza, il paradigma anticipatorio per poter effettuare le predizioni ha bisogno di un modello interno e tale modello è rappresentato nel nostro caso da una mappa dell'ambiente. L'operazione di costruzione della mappa è svolta appunto dall'applicazione MapBuilding. Lo schema architetturale del suo modulo di elaborazione è visibile in figura 4.1. La struttura dati è, ovviamente, la mappa.

Per scelta progettuale, si è scelto di utilizzare le Griglie Regolari, in quanto sono tra le più usate e permettono una valida rappresentazione delle caratteristiche del mondo (feature-based), essenziale e poco costosa a livello computazionale, che si adatta al tipo di risposta sensoriale ottenuta con l'uso dei sonar. La Griglia Regolare è stata implementata tramite una matrice di interi $\mathbb{N} \times \mathbb{N}$. La struttura dati condivisa tra l'interfaccia utente e il thread

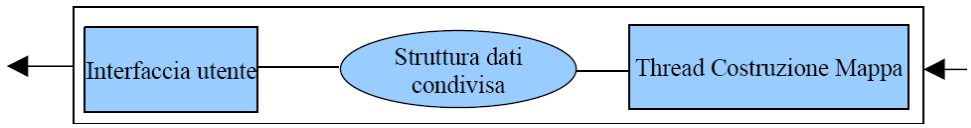

Figura 4.1: Schema dell'applicazione MapBuilding

di costruzione della mappa, quindi, non è altro che questa matrice; si accede a quest'ultima in mutua esclusione tramite un pattern Proxy che verrà descritto in dettagli nel par 4.5.1.

Un'applicazione che deve costruire una mappa è composta generalmente da un algoritmo di esplorazione, da uno di localizzazione e da uno che prende i dati dai sensori e li usa per costruire effettivamente la mappa.

Nel presente lavoro di tesi non è stato implementato un algoritmo di esplorazione, ma la scelta è stata quella di teleguidare il robot tramite l'interfaccia grafica; il vantaggio principale di questo approccio è stato quello di riuscire a modellare al meglio l'ambiente circostante facendo passare il robot più e più volte nei punti dove la modellazione non era riuscita perfettamente. Per questo motivo l'interfaccia grafica è stata dotata di una sorta di joystick virtuale minimale.

Essendo la procedura di esplorazione manuale, non è necessario implementare un sofisticato algoritmo di localizzazione. Infatti, la localizzazione, in questo caso, è utilizzata esclusivamente dall'algoritmo di costruzione della mappa per individuare la posizione del robot e quindi sapere quali sono le celle della mappa da aggiornare. L'odometria è quindi utilizzata ad ogni traslazione per localizzare il robot. Per quanto riguarda le rotazioni, avendo utilizzato le griglie regolari, è sufficiente far compiere al robot dei movimenti di 90°. Tramite interfaccia grafica è possibile inoltre affinare la procedu-

ra di rotazione tramite l'apposito pulsante "Fine", in modo da correggere manualmente eventuali errori meccanici di movimento.

Per quanto riguarda l'algoritmo per la costruzione della mappa, è stato scelto l'HIMM, che, come nel caso delle griglie regolari, è la scelta più comune tra quelle proposte dallo stato dell'arte grazie alle sue caratteristiche di velocità e leggerezza.

4.1.1 Implementazione dell'algoritmo HIMM

Per prima cosa l'algoritmo acquisisce la posizione attuale del robot nella mappa, tale informazione verrà passata manualmente all'algoritmo nell'inizializzazione e in seguito, calcolata tramite odometria. Successivamente acquisisce le letture dai sonar. In conseguenza del fatto che il mondo è rappresentato tramite griglie regolari, i valori ottenuti dai sonar, che si riferiscono all'intera "vista" a 360° del robot, possono essere approssimati nelle seguenti otto direzioni: le quattro ortogonali Nord, Sud, Est, Ovest e le restanti quattro diagonali Nord-Est, Nord-Ovest, Sud-Est, Sud-Ovest. Tale approssimazione avviene calcolando il valor medio delle risposte sensoriali ottenute dai sonar che sono collocati in direzione di questi punti cardinali, ovviamente relativi al robot.

Infine l'algoritmo esegue gli aggiornamenti sulle celle corrette, basandosi sulla propria posizione nella mappa e sul verso in cui è ruotato rispetto alla mappa stessa e ai valori medi calcolati.

Per il calcolo dei valori con cui aggiornare la casella è stato fatto uso dell'operatore GRO. Questo perché permette di rendere visibili oggetti che l'HIMM "puro", difficilmente o dopo ripetute iterazioni, riesce a percepire come ostacoli, come esposto in maniera più approfondita nel cap 3.3.5.

4.1.2 L'applicazione MapBuilding

L'interfaccia grafica è visibile in figura 4.2. Nella parte superiore sono presenti diversi pulsanti:

Connect connette l'applicazione al robot in modo da poter ricevere i dati dai sensori e spedire i comandi al robot stesso;

Set permette di inserire i parametri relativi alla dimensione della griglia, stabilendo così la dimensione del mondo e l'ampiezza di ogni singola cella della griglia;

StartPoint consente di decidere la posizione iniziale e la rotazione del robot nella griglia, dopo che questa è stata creata tramite il precedente pulsante *Set*;

Build inizia l'esecuzione dell'algoritmo che esegue la costruzione della mappa;

Save permette il salvataggio della mappa sul database in modo che possa essere successivamente utilizzata dall'applicazione di navigazione.

L'algoritmo HIMM aggiorna le celle solo quando il robot si muove in avanti; questo perché quando durante le rotazioni il robot è fermo e rileggerebbe le stesse celle solo con sensori diversi. In più, come spiegato in dettaglio nel paragrafo l'HIMM ha delle prestazioni migliori quando il robot è in movimento, possibilmente anche in maniera piuttosto rapida.

Nella parte inferiore sono presenti i pulsanti relativi alla teleoperazione del robot, come mostrato in figura 4.2:

Avanti fa avanzare il robot;

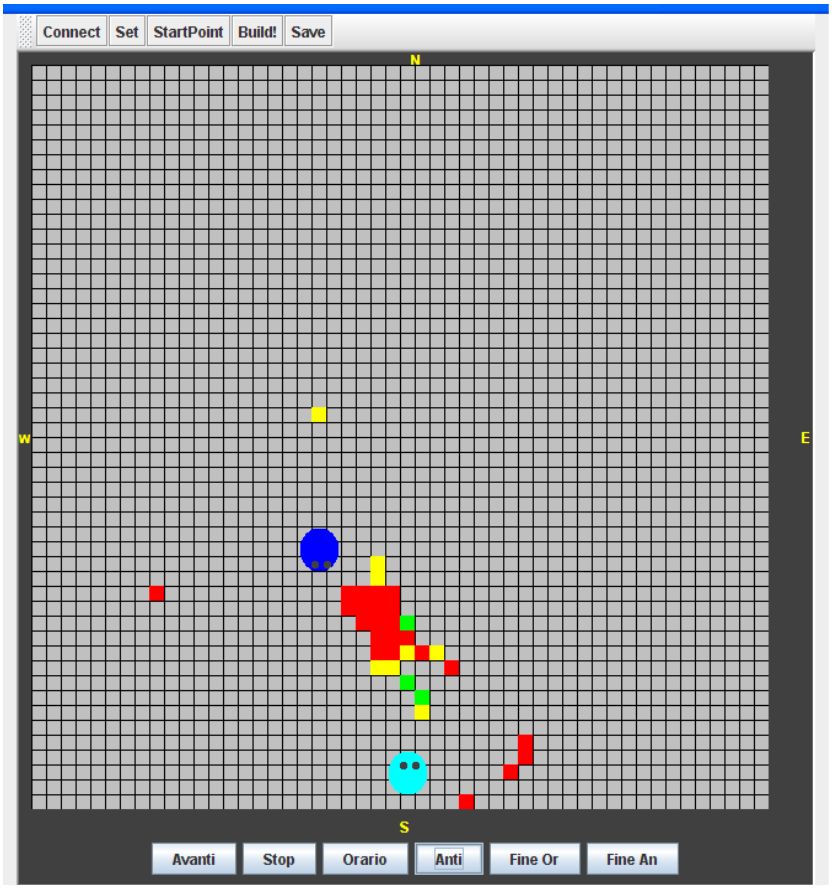

Figura 4.2: Costruzione della mappa con MapBuilding

Stop ferma il robot;

Orario ruota il robot di 90° in senso orario;

Anti ruota il robot di 90° in senso antiorario;

Fine Or esegue piccole rotazioni di 3° in senso orario, utile per fare correzioni in presenza di eventuali imprecisioni nelle rotazioni più ampie;

Fine An esegue piccole rotazioni di 3° in senso antiorario.

Nella rappresentazione grafica di figura 4.2 si vede l'algoritmo HIMM in azione. Il cerchio celeste chiaro segna la cella dalla quale è partito il robot e gli "occhi" indicano l'orientamento iniziale nel quale era ruotato rispetto alla mappa. Il cerchio blu indica la posizione attuale del robot e gli "occhi" indicano la sua orientazione. I quadrati della griglia sono di colori diversi: ad ogni colore è associato un range di valori assegnati dall'HIMM ad ogni cella. Come schematizzato nella seguente tabella, una cella di colore rosso rappresenta un valore compreso tra 13 e 15, questo vuol dire che l'algoritmo considera tale cella come occupata, il giallo è un valore intermedio che va da 7 a 12 ed esprime incertezza, il verde va da 1 a 6 ed indica una quasi certezza che non ci siano ostacoli ed infine il grigio rappresenta lo 0, quindi assenza di ostacoli.

Colore Cella	Range	Significato
Rosso	13-15	Occupato
Giallo	7-12	Incertezza
Verde	1-6	Molto probabilmente libero
Grigio	0	Libero

Normalizzazione della mappa e salvataggio

Prima di memorizzare la mappa nel database, l'applicazione effettua 2 operazioni preliminari di "normalizzazione": la sogliatura e un'operazione di "tappa-buchi".

Durante la sogliatura, l'algoritmo confronta i valori di ogni cella della mappa con una certa soglia, se maggiore lo eguaglia al valore massimo, se è inferiore, lo eguaglia a 0.

L'operazione tappa-buchi consiste nel chiudere quei piccoli "corridoi" di

larghezza inferiore al diametro del robot. In sostanza, tutti i passaggi troppo stretti per il robot vengono considerati come veri e propri ostacoli. Questa procedura permette quindi all'algoritmo di navigazione di escludere a priori i percorsi impraticabili dal robot. Senza tale accorgimento si sarebbe compromessa l'efficienza del sistema, poiché il robot, una volta giunto in prossimità di un passaggio troppo stretto per le sue dimensioni, avrebbe dovuto creare un nuovo path.

Questa operazione si sarebbe potuta evitare se il lato di ogni cella fosse grande quantomeno il diametro del robot, ma questa soluzione porta ovviamente ad una modellazione del mondo più grossolana e meno precisa.

In figura 4.3 è mostrata una mappa sottoposta a normalizzazione.

E' evidente come siano scomparse le celle di colore verde e giallo e come siano stati eliminati i buchi presenti nella modellazione degli ostacoli. Dopo l'operazione di normalizzazione si procede col salvataggio della mappa sul database. Nella figura 4.4 sono mostrati gli schemi relazionali della tabella delle mappe e delle celle.[AGO97]

Come si può vedere nello schema relazionale ogni cella fa riferimento alla propria mappa tramite la chiave esterna ID. In più ognuna di esse ha un valore, che è quello assegnato dall'algoritmo HIMM e le sue coordinate x, y all'interno della mappa.

4.2 Sistema di navigazione con EP

Lo schema architetturale del sistema di navigazione con Predizione è mostrato in figura 4.5.

Come detto in precedenza, il modello interno è costituito dalla mappa che rappresenta il mondo.

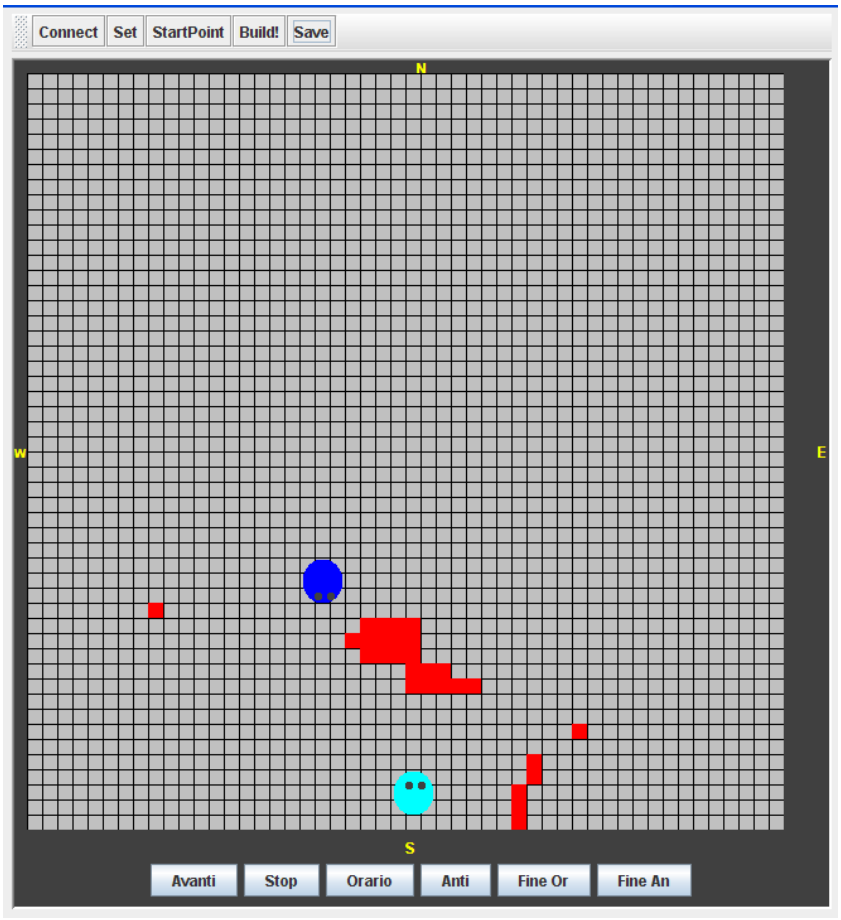

Figura 4.3: Mappa dopo l'operazione di normalizzazione

Il blocco di predizione deve ricavare dal modello interno, una previsione sui dati di lettura. Questo avviene prevedendo la nuova posizione del robot in base al movimento che sta effettuando e al tempo in cui deve avvenire il confronto sui dati reali; una volta prevista la nuova posizione, si ricava la distanza dall'ostacolo più vicino in ogni direzione. Un'operazione da effettuare è la conversione da numero di celle, dalla prima occupata, a distanza nel mondo reale: questa è una semplice moltiplicazione tra il numero di celle e la dimensione della cella nel mondo reale.

Il blocco di confronto deve verificare i dati predetti con i dati reali pro-

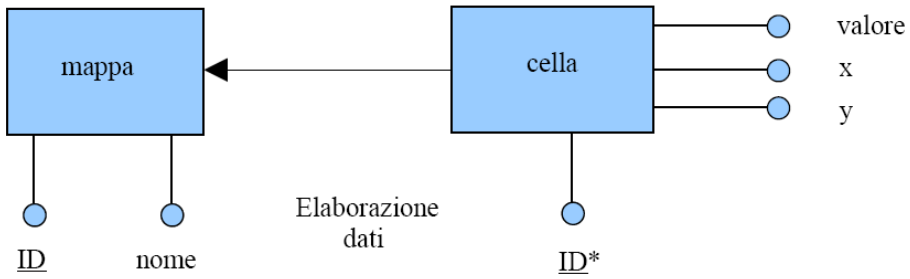

Figura 4.4: Schema relazionale della mappa

venienti dai sensori. Si fa uso di un valore soglia Θ in modo da stabilire un certo margine di errore nella predizione per ovviare ai seguenti problemi:

- errore nella lettura dei sonar;
- discretizzazione dei dati dei sonar nella costruzione della mappa: la distanza di un oggetto dal robot è un numero reale; questa distanza viene elaborata, ottenendo la posizione della cella in cui è contenuto l'ostacolo. Quando si fa l'operazione inversa, però, non si può sapere qual'è la distanza esatta, ma un range di valori entro la quale è posta la distanza dell'ostacolo.

Se la differenza tra la distanza predetta dell'ostacolo e quella calcolata è maggiore di Θ, significa che quell'ostacolo non è modellato e a quel punto il sistema esegue tutte le procedure per convertire le letture dei sonar in valori relativi alle coordinate di mappa, aggiornare quest'ultima e creare un nuovo percorso ottimale. In caso contrario, ossia nel caso in cui la differenza risulti minore del valore soglia, vengono generati i comandi di movimento del robot per proseguire nel percorso di raggiungimento del goal.

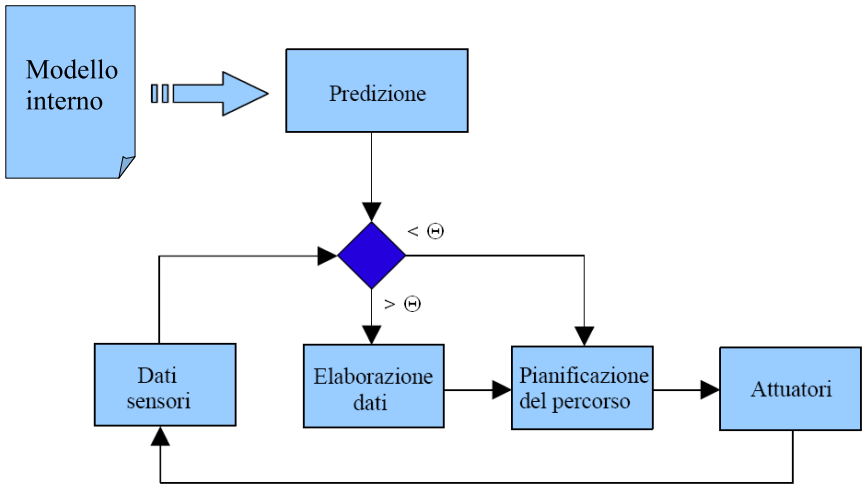

Figura 4.5: Architettura del sistema di navigazione con predizione

I dati provenienti dai sonar non sono quindi processati prima del confronto, perché il modulo di predizione, dall'analisi del modello interno, fornisce dei valori conformi ai dati dei sonar. Questo permette di anticipare il calcolo più pesante e quindi di rendere il robot più reattivo in presenza di ostacoli già modellati.

Per la generazione del percorso ottimale dalla cella iniziale al goal si è deciso di utilizzare l'algoritmo A*.

Per la localizzazione si è usata l'odometria come già per la costruzione della mappa. Per quanto riguarda la pianificazione del movimento, si è assunto che il robot assumesse solo quattro orientazioni e quindi di limitare i suoi movimenti rotatori a 90°. Questa decisione è dovuta al fatto che è accettabile pensare che il robot si possa muovere solo nelle 4 celle adiacenti e non sulle diagonali.

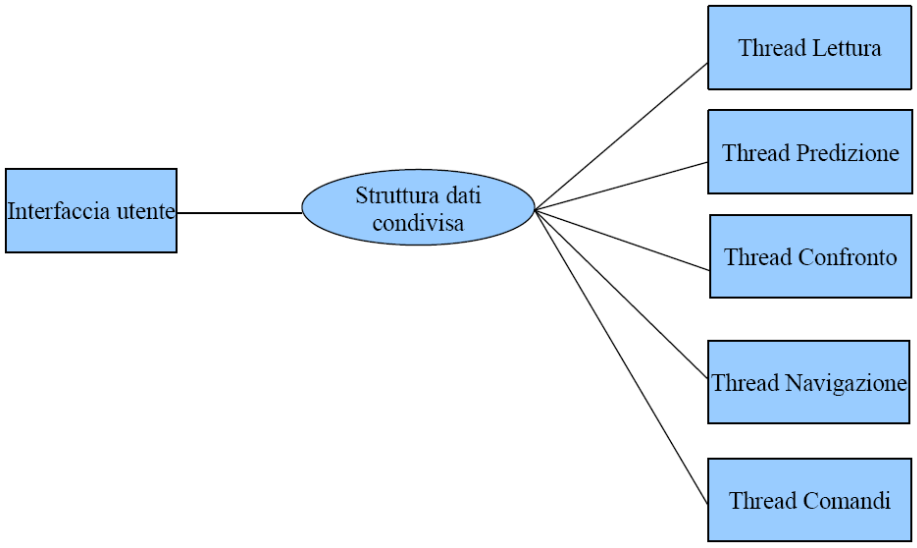

Figura 4.6: Schema applicazione Navigation con Predizione

4.2.1 Applicazione Navigation con EP

Lo schema dell'applicazione è visibile in figura 4.6.

Il thread di predizione lanciato ad intervalli regolari predice i valori dalla mappa e attende l'intervallo successivo: a questo punto accede, in mutua esclusione, ad una variabile condivisa, in cui copia la previsione fatta precedentemente, notifica al thread di confronto, tramite un opportuno monitor, che sono disponibili nuovi dati ed esegue una nuova predizione. Di seguito si può vedere parte del codice che esegue questa operazione:

```
//Variabile dove si accumulano i dati predetti
double[] prediction = new double[8];

...

//Meccanismo di accesso in mutua esclusione
//alla variabile condivisa "match.prediction"
//match è il thread che esegue il confronto
//fra i dati reali e quelli predetti
```

```
//m è il monitor

if (sinc && emer.status == ThreadEmergency.FREE){
    match.prediction =prediction;
    m.riparti();
}
```

Così facendo si è implementato qualcosa di simile al double buffer della grafica: si crea un'immagine (predizione) in memoria; alla successiva richiesta di dati viene passata l'immagine (predizione) già in memoria e nel frattempo se ne crea un'altra. Questo meccanismo permette di essere più veloci e in questo caso di poter accedere alla variabile in mutua esclusione ma senza dover aspettare che l'altro thread concorrente abbia finito di usarla. Infatti il Thread di predizione accede alla variabile prima di risvegliare il thread di confronto e poi vi riaccede al risveglio successivo. Nel frattempo la variabile condivisa può essere letta a piacimento dal thread di confronto mentre il thread di predizione salva la nuova predizione su una variabile locale che solo al passo successivo copierà nella variabile condivisa.

Nel caso in cui il confronto risulti negativo, il thread genera comandi di emergenza fermando il robot e attende un tempo prefissato per verificare che il nuovo ostacolo non sia in movimento, evitando così di inserire oggetti in movimento, e quindi temporanei, nella mappa. Dopodiché esegue nuovamente il confronto con una nuova lettura e , se l'ostacolo non modellato è sempre presente, fa aggiornare la mappa al Thread di emergenza, che provvederà a calcolare un nuovo path.

Di seguito è riportato il codice che effettua il confronto:

```
if(( val + OFFSET < prediction[0]))
//dove "prediction" è il dato predetto,
```

```
//"offset" è un margine di errore concesso
//alla lettura reale e "val" è il valore
//letto dai sensori
//Tutti i valori sono definiti in metri
//e il tipo delle variabili è Double.
```

Per la generazione di un percorso ottimale è stato scelto di implementare l'algoritmo A*. Per sfruttare le potenzialità di questo algoritmo, la mappa viene trasformata in un grafo. Tale procedimento è esposto nel paragrafo successivo.

4.2.2 Trasformazione della mappa in un grafo

La trasformazione della mappa in grafo avviene quando il modulo pianificatore deve generare il minimo percorso dal punto di partenza al punto di arrivo (goal).

Il grafo è stato implementato tramite una matrice di adiacenza.

Si definisce matrice di adiacenza A di un grafo $G(V, E)$, la matrice $|V| \times |V|$ tale che:

$$\begin{cases} k & \text{se}(i,j) \in E(G) \\ 0 & \text{se}(i,j) \notin E(G). \end{cases}$$

con $k \in \mathbb{N}$. L'intero k rappresenta il peso relativo dell'arco. Tale peso è ottenuto tramite il calcolo della Manhattan Distance tra il nodo in esame e il goal. La Manhattan Distance è una metrica in cui la distanza tra due punti viene calcolata come la somma delle differenze (in valore assoluto) delle loro coordinate.

Formalmente, si può definire Manhattan distance tra due punti nello spazio euclideo, con un fissato sistema di coordinate cartesiane, la somma delle

lunghezze delle proiezioni sugli assi cartesiani dei segmenti che congiungono i due punti.

Per esempio, nel piano, la distanza L_1 tra due punti P_1 di coordinate (x_1, y_1) e il punto P_2 di coordinate (x_2, y_2) è

$$L_1(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$$

La distanza L_1 rappresenta la minore distanza tra due punti in un fissato sistema di coordinate cartesiane.

Ogni cella della mappa diventa, quindi, un nodo del grafo e ha quattro archi (ad eccezione delle celle perimetrali).

In pratica, con la procedura di trasformazione, si ottiene un grafo pesato, in cui i valori sugli archi rappresentano la minima distanza tra il nodo in esame e il nodo goal. Un arco con peso 0 denota, invece, un percorso tra i due nodi impraticabile ed è come se non esistesse, per questo non verrà preso in considerazione dall'algoritmo per la generazione del cammino minimo. Una cella della mappa che risulta occupata sarà nel grafo un nodo avente tutti e quattro gli archi con peso nullo.

Tornando all'implementazione del grafo tramite matrice di adiacenza, si ha che questa ha dimensione quadratica rispetto a quella della mappa. Questa rappresentazione facilita l'individuazione dei vertici adiacenti, proprietà fondamentale per la generazione del percorso.

La principali rappresentazioni di un grafo sono due: tramite liste di adiacenza o matrici di adiacenza. Queste ultime sono preferibili in presenza di grafi densi, ossia quando il numero degli archi è molto maggiore di quello dei nodi, tipicamente $|E| \sim |V|^2$. Il vantaggio che se ne trae è un più rapido accesso ai dati.

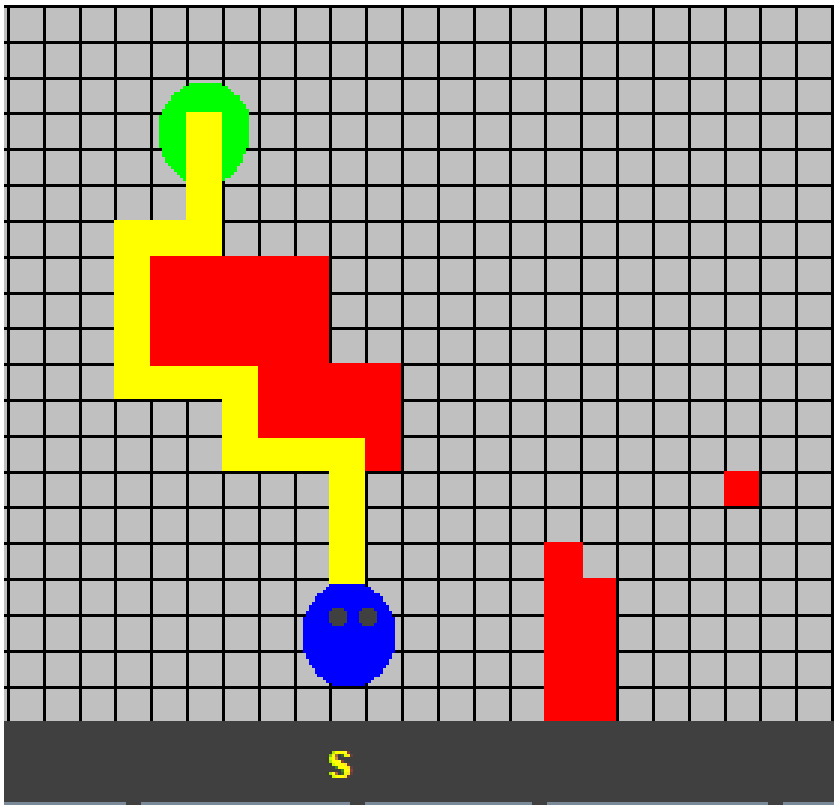

Figura 4.7: Percorso creato con A*

4.2.3 Creazione del percorso ottimale

Come algoritmo per la creazione del percorso è stato scelto A*. È stato preferito al D* perché più leggero dal punto di vista computazionale, in quanto non bisogna creare tutti i percorsi che portano al goal con relativa occupazione di memoria oltre al tempo per realizzarli e poi si evita di dovere aggiornare una quantità esagerata di percorsi una volta che si trova un nuovo ostacolo. Il D* sarà più utile in ambienti con pochi ostacoli ancora da modellare o dove i percorsi sono molto lunghi, per i quali, ricostruire nuovi path da zero è più costoso che aggiornarne diversi dinamicamente. In figura 4.7 è mostrato un percorso ottimale che va dal punto di partenza fino all'obiettivo.

4.2.4 Algoritmo di pianificazione e localizzazione

Questo algoritmo è implementato nel thread di navigazione. Il thread per prima cosa verifica se si è raggiunto il nodo successivo del grafo; in caso positivo aggiorna il nodo da raggiungere. In seguito controlla, a seconda dell'orientamento del robot, se deve eseguire una rotazione per raggiungere il nodo successivo: questo controllo si basa sugli indici della cella nel quale è localizzato il robot e su quelli del nodo successivo. Una volta eseguita l'eventuale rotazione, parte per raggiungere il nodo successivo.

La localizzazione è composta dalla posizione sulla mappa e dall'orientamento. Quest'ultimo viene mantenuto tenendo conto delle rotazioni di 90° effettuate dal robot e del suo orientamento iniziale. La posizione viene calcolata partendo dalla posizione precedente e sommandoci lo spazio percorso nel frattempo: sarà la differenza tra l'ultima lettura dello spazio percorso dell'odometria e il valore attuale dello stesso. Questa localizzazione non sarà esatta a causa degli errori dell'odometria per quanto concerne la posizione e per problemi dovuti alla gestione delle rotazioni per quanto riguarda l'orientamento. Per quest'ultimo problema, non si sa con certezza se la rotazione effettivamente eseguita dal robot sia di 90° e non è opportuno controllare l'odometria, in quanto gli oscilloscopi che vengono usati possono avere problemi di stabilità quando il robot è in movimento. Quindi è stata usata una localizzazione incerta per questa applicazione.

4.2.5 Obstacle Avoidance

Questo algoritmo è stato implementato nel thread di emergenza. Nel caso in cui la predizione dei dati risultasse errata, si deve eseguire la procedura di obstacle avoidance. Il thread recupera i dati che non coincidono con quelli

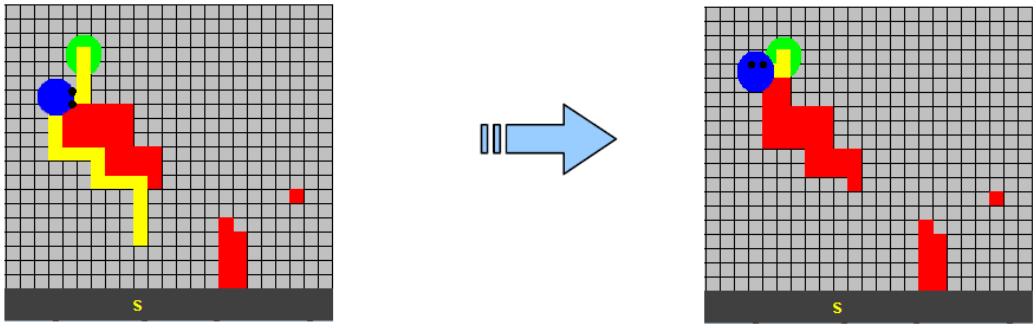

Figura 4.8: Aggiornamento della mappa con un nuovo ostacolo

della mappa, tipicamente un nuovo ostacolo, individua le coordinate relative alle celle della mappa e aggiorna i rispettivi valori. Nell'inserimento di un nuovo ostacolo si è preferito aggiornare anche le celle contigue, considerando in tal modo le sue dimensioni più grandi rispetto a quelle percepite. In tal modo, si ottiene un margine di sicurezza nei confronti di quel particolare ostacolo e ci si cautela verso eventuali errori di lettura dei sonar. Un esempio di aggiornamento della mappa è visibile in figura 4.8.

Aggiornata la mappa, viene riapplicato l'algoritmo di generazione del percorso ottimale sul nuovo grafo e si generano i comandi per raggiungere il nuovo nodo del grafo.

4.2.6 Funzionamento di Navigation

L'applicazione Navigation permette di scegliere quale mappa usare fra quelle contenute all'interno del database. Una volta scelta visualizzerà la griglia come in figura 4.9. A questo punto, bisogna connettersi al robot per cominciare a ricevere i dati dai sonar e poter spedire i comandi al robot. In seguito bisogna indicare il punto di partenza del robot (sarà visualizzato con cerchio blu) e la cella del goal (indicata con cerchio verde). Una volta eseguite queste

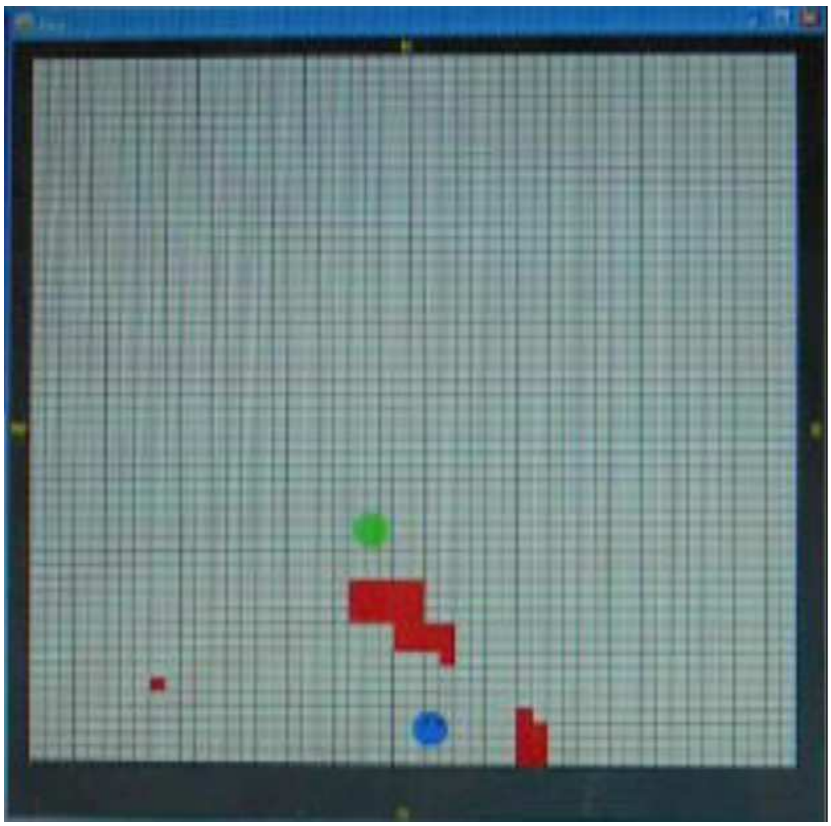

Figura 4.9: Apertura della mappa dal database e inseriti posizione iniziale e goal

operazioni basta cliccare sul pulsante "Go!" per avviare l'algoritmo A*, che provvederà a creare il path. Appena il percorso sarà disponibile, il robot inizierà a seguirlo. Il processo si arresta o quando il robot raggiunge il goal, ed in questo caso si può stabilire il nuovo goal e ripartire, o quando si rende conto che la cella dell'obiettivo non è raggiungibile dall'attuale posizione a causa degli ostacoli presenti sulla mappa. Quest'ultima indicazione è data dall'algoritmo A* quando non riesce a stabilire un percorso dal nodo iniziale al goal.

Figura 4.10: Console dei comandi

4.3 Sistema di navigazione senza EP

Nel corso del presente lavoro di tesi, è stato implementato anche un sistema di navigazione che non fa uso di predizione, in modo da poter confrontare i rispettivi comportamenti. Lo schema architetturale del sistema di navigazione senza predizione è mostrato in figura 4.11. L'architettura è strutturata come un ciclo di operazione sequenziali: lettura dei dati dai sensori, elaborazione degli stessi per identificare l'eventuale presenza di ostacoli non modellati, procedendo di conseguenza all'aggiornamento della mappa interna, pianificazione del movimento successivo per raggiungere il sotto-obiettivo, passaggio dei comandi relativi al robot.

Come nel caso del sistema di navigazione con predizione, si è scelto di uti-

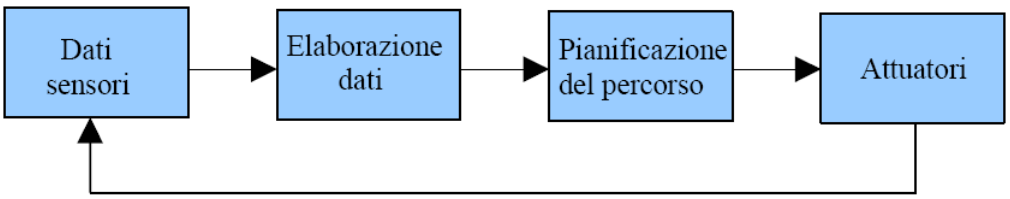

Figura 4.11: Architettura del sistema di navigazione senza predizione

lizzare una mappa e di usare l'algoritmo A* per generare il percorso minimo dalla cella iniziale al goal.

4.3.1 Applicazione Navigation senza EP

Lo schema dell'applicazione è visibile in figura 4.12. Essa è composta da 4 thread più l'interfaccia utente che condividono una stessa struttura dati; questa non è altro che la mappa la quale, come già nel MapBuilding, è implementata tramite il pattern Proxy per l'accesso in mutua esclusione tra i vari thread.

L'interfaccia utente permette di scegliere la mappa che modella il mondo nel quale il robot agirà, stabilire la posizione iniziale del robot e la sua orientazione, assegnare un goal e far eseguire l'algoritmo di navigazione.

Il thread di navigazione rappresenta il modulo pianificatore. Durante l'inizializzazione trasforma la mappa in un grafo e crea il percorso per raggiungere l'obiettivo tramite l'algoritmo A*. Successivamente, in base ai dati odometrici, genera i comandi motori da effettuare per raggiungere il nodo successivo del path. Tali comandi verranno utilizzati dal Thread dei Comandi.

Il thread di emergenza analizza i dati dai sensori e, se riscontra un ostacolo, lo confronta sulla mappa. Se l'ostacolo non è modellato, effettua tutte le operazioni necessarie per aggiornare la mappa, trasformarla in un grafo e

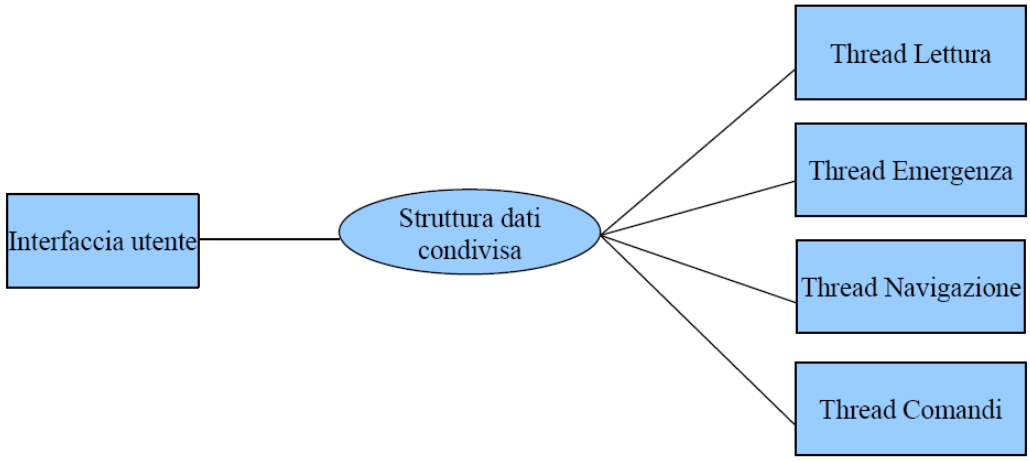

Figura 4.12: Schema applicazione Navigation senza Predizione

creare il nuovo percorso dalla locazione attuale del robot fino al goal, con le stesse modalità con cui le esegue il sistema con predizione.

La differenza principale tra i due sistemi di navigazione si ha quando vengono rilevati degli ostacoli. Il sistema senza predizione deve trasformare i dati sensoriali in coordinate della mappa per verificare se l'ostacolo risulta già modellato, in caso contrario dovrà aggiornare la mappa. Nel sistema con EP, queste operazioni sono anticipate dal modulo di predizione, dovrà effettuare soltanto una differenza tra numeri in virgola mobile e confrontarla con un valore soglia.

4.4 Struttura generale delle applicazioni

La struttura generale delle due applicazioni è mostrata in figura 4.13 : è costituita da tre blocchi assemblati secondo il paradigma di parallelismo pipeline. I blocchi di lettura dati dal robot e quello di invio comandi allo stesso sono comuni ai due applicativi e servono per disaccoppiare il modulo di elaborazione, in cui sono implementati tutti gli algoritmi, dal robot uti-

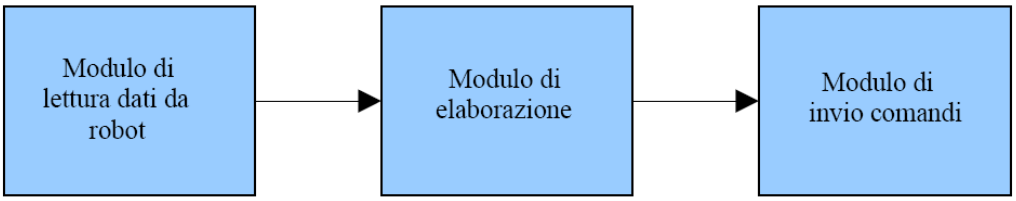

Figura 4.13: Schema Pipeline delle due applicazioni

lizzato. In questo modo, il modulo di elaborazione è indipendente dal robot utilizzato. L'unica limitazione che è stata imposta alla tipologia del robot è quella di avere sonar disposti in modo da permettere una copertura a 360°. I moduli di scrittura e lettura sono realizzati su due thread separati, proprio per sfruttare il parallelismo. Le due applicazioni differiscono, appunto, per il modulo di elaborazione.

4.4.1 Paradigma Pipeline

Una pipeline consiste di una catena di stadi di elaborazione (processi, thread, coroutines ecc.), posti in modo che l'uscita di ogni stadio è l'ingresso del prossimo [Van06]. Generalmente ci sono dei buffer tra 2 elementi per poter memorizzare dati da elaborare nel caso che uno stadio precedente sia più veloce del successore. Le informazioni che fluiscono in queste pipelines sono uno stream di dati, byte o bits. Nel nostro caso gli stadi sono thread e lo stream è l'insieme dei dati dei sensori, che, trasformati, diventano comandi per il robot.

Con un paradigma sequenziale, l'elaborazione di un dato attraversa tutti gli stadi e soltanto quando ogni calcolo su di esso è terminato può essere processato un dato successivo.

Facendo uso del paradigma di parallelizzazione pipeline, invece, non appe-

na un modulo ha terminato l'elaborazione di un dato, passa immediatamente al successivo. Così facendo non si migliorano le prestazioni per l'elaborazione di un singolo dato, perché il tempo di elaborazione (latenza) rimane lo stesso, ma si migliora il tempo complessivo di elaborazione, perché passa dalla somma delle latenze di tutti i dati, al massimo tra i tempi di servizio.

Il tempo di servizio è il tempo necessario ad elaborare un singolo dato da parte di uno stadio, conseguentemente il tempo di servizio della pipeline è il maggiore tra i tempi di servizio dei vari stadi. Questo perché il tempo trascorso tra il termine dell'elaborazione di 2 dati consecutivi è proprio il tempo di servizio.

Usando questo paradigma di parallelizzazione il sistema potrà fare una lettura dai sensori e passarla al modulo di elaborazione e, mentre questo elabora i dati passati, fare un'altra lettura. Allo stesso modo, mentre il modulo di elaborazione processa i dati di lettura, il modulo di spedizione comando spedisce direttive al robot.

In questo caso il modulo più lento sarà quello di elaborazione, quindi il tempo di servizio della pipeline sarà il tempo di servizio del modulo di elaborazione.

4.5 Modulo di lettura

Il modulo di lettura è fortemente dipendente dal robot utilizzato. Nella nostra implementazione, la prima operazione è quella di collegarsi al robot tramite protocollo telnet e impostare gli oggetti per poter comunicare con il robot. Successivamente, ad intervalli regolari, interroga il robot e recupera i dati dai vari sensori. In questo caso i dati recuperati sono quelli dei sonar e quelli dell'odometria.

I dati letti devono essere condivisi tra il thread di lettura e i thread del modulo di elaborazione, che li usa per generare i comandi da mandare al robot. Per far questo usa un oggetto condiviso tra i vari thread. Per poter accedere a questo oggetto in mutua esclusione, quest'ultimo è stato implementato con il pattern Proxy.

4.5.1 Pattern Proxy

Un design pattern è la soluzione generica a problemi che si ripetono nel tempo, cioè, una soluzione già testata che serve a risolvere una serie di problematiche di routine. Infatti, Christopher Alexander dice che:"Ogni pattern descrive un problema che si ripete più e più volte nel nostro ambiente, descrive poi il nucleo della soluzione del problema, in modo tale che si possa usare la soluzione un milione di volte, senza mai applicarla nella stessa maniera".[GHJV02]

Questi pattern sono molto usati in ingegneria del software per risolvere problemi comuni. Il pattern Proxy è utile in tutte quelle occasioni in cui non vogliamo accedere direttamente ad un oggetto, ma averne un altro che ne faccia da intermediario. Infatti, il pattern prende il nome dai proxy delle reti di computer che fanno da intermediari tra la sottorete ed internet.

Comunque esistono varie situazioni nelle quali il pattern Proxy è applicabile:

- un proxy remoto fornisce un rappresentante locale per un oggetto in un diverso spazio di indirizzamento;
- un proxy virtuale gestisce la creazione su richiesta di oggetti computazionalmente costosi;

- un proxy di protezione controlla l'accesso a un oggetto. Questo tipo di proxy si rivela utile quando possono essere definiti diritti di accesso diversi per gli oggetti;
- un riferimento intelligente sostituisce un puntatore puro ad un oggetto, consentendo l'esecuzione di attività aggiuntive quando si accede all'oggetto referenziato. Alcuni utilizzi tipici sono:
  - ♢ il conteggio del numero dei riferimenti all'oggetto reale, in modo da gestire automaticamente la sua distruzione quando quando non ci sono più riferimenti attivi;
  - ♢ il caricamento di un oggetto persistente in memoria quando viene referenziato per la prima volta;
  - ♢ la verifica che l'oggetto rappresentato sia riservato (locked) quando si richiede di accedervi, in modo che nessun altro oggetto possa modificarlo (mutua esclusione).

Nel nostro caso specifico, usare questo design pattern ci serve per accedere in mutua esclusione ad una HashMap dove sono inseriti tutti i dati provenienti dai sensori del robot.

In figura 4.14 è mostrata la struttura del pattern Proxy.

L'oggetto Proxy mantiene un riferimento che gli consente di accedere all'oggetto rappresentato di tipo RealSubject. Un Proxy può accedere ad un Subject se le interfacce di Subject e RealSubject sono identiche. La classe Proxy fornisce un'interfaccia identica a quella definita da Subject, consentendo, quindi, al Proxy, di agire come un sostituto dell'oggetto destinazione. Il Proxy controlla l'accesso all'oggetto rappresentato e può essere responsabile della sua creazione ed eliminazione. I Proxy remoti sono responsabili

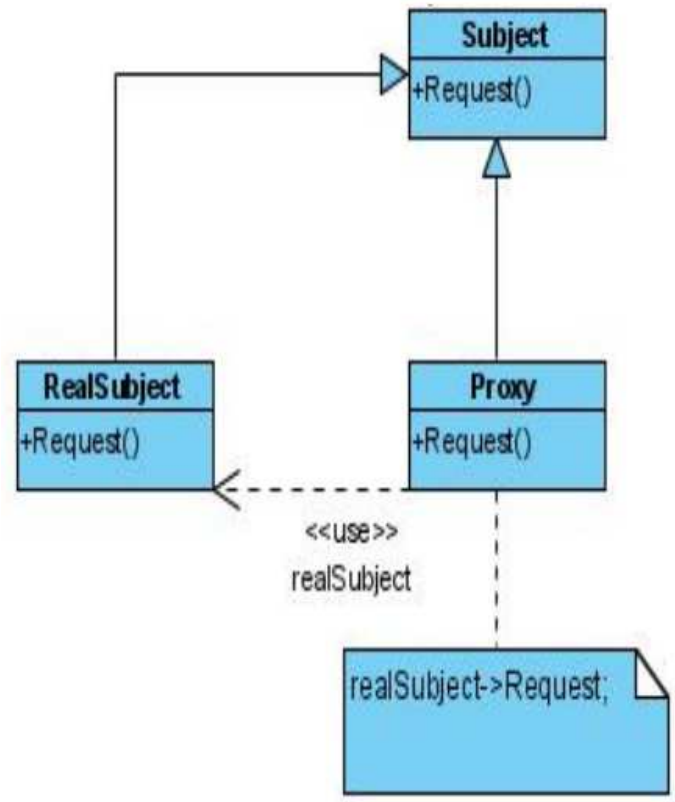

Figura 4.14: Struttura del pattern Proxy

della codifica di una richiesta e dei suoi argomenti, nonché dell'invio della richiesta codificata all'oggetto rappresentato, posto in un diverso spazio di indirizzamento. I Proxy virtuali possono memorizzare informazioni addizionali sull'oggetto rappresentato allo scopo di posticipare l'accesso a tale oggetto. I Proxy di protezione verificano che il chiamante abbia i permessi di accesso necessari per poter effettuare una richiesta all'oggetto rappresentato. La classe Subject definisce l'interfaccia comune per le classi RealSubject e Proxy, rendendo possibile l'utilizzo del Proxy in tutte le situazioni in cui è

previsto l'utilizzo di RealSubject. La classe RealSubject caratterizza l'oggetto reale rappresentato dal Proxy. Un Proxy trasferisce le richieste ricevute a RealSubject quando è opportuno, in base alla specifica tipologia di proxy utilizzata.

4.6 Modulo di scrittura

Il modulo di scrittura non fa altro che attendere un comando dal modulo di elaborazione e spedirlo al robot. Per far questo esegue un controllo ad intervalli regolari in attesa di avere un dato da processare. I comandi al modulo vengono passati attraverso due variabili accedibili in mutua esclusione tramite l'utilizzo del costrutto synchronized della libreria standard di Java. Una delle due variabili (*emer*) è utilizzata per inviare comandi in caso di emergenza, cioè nel caso in cui ci sia un reale rischio che il robot stia per urtare un ostacolo. L'altra (*com*) è utilizzata per l'invio di comandi generici.

La sicurezza è garantita dal fatto che il thread prima di eseguire un comando di movimento "normale" controlla se è presente un comando in *emer*. Se c'è, esegue l'emergenza e "pulisce" entrambe le variabili, in caso contrario, viene esegue il comando indicato nella variabile *com* e lo elimina per non eseguirlo nel ciclo successivo.

Quindi un comando di emergenza acquisisce priorità su un comando normale, che anzi, non viene neppure eseguito se è in concomitanza con uno dell'altro tipo.

Capitolo 5

Robot B21r

Il lavoro di tesi è stato svolto sul robot **B21r** prodotto dalla **iRobot RWI** *(Real World Interface, Inc.)*. Il robot al suo interno monta un PC Pentium 200MHz ed è alimentato da 4 batterie 12V/31Ah. L'applicazione sviluppata è stata pensata per essere eseguita su di un computer remoto, per questo è stata creata una rete LAN wireless tramite un ponte Linksys installato sul robot.

I sensori del robot che sono stati utilizzati durante il lavoro di tesi sono :

- 32 Sensori ad ultrasuoni nella base
- 32 Sensori di contatto nella base
- 24 Sensori ad ultrasuoni nella parte superiore
- 24 Sensori di contatto nella parte superiore
- sensori odometrici

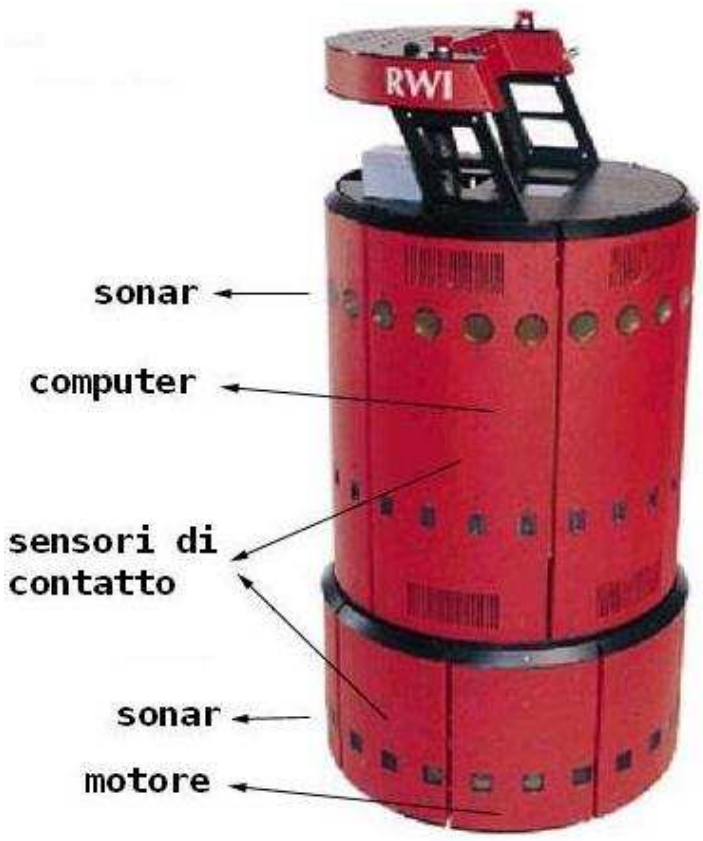

Figura 5.1: Robot b21

5.1 Il software integrato : Mobility

Il software integrato del robot è costituito da un toolkit distribuito e orientato agli oggetti per la costruzione di software per il controllo di robot singoli o multipli.

Come mostrato nella Figura 5.2, Mobility definisce il *Mobility Robot Object Model* usando il *Common Object Request Broker Architecture* (CORBA) 2.X standard *Interface Definition Language* (IDL). In questo modo, seguendo questo standard, Mobility supporta molti linguaggi su diverse piattaforme. Il *Robot Object Model* definisce un sistema robotico come un insieme di og-

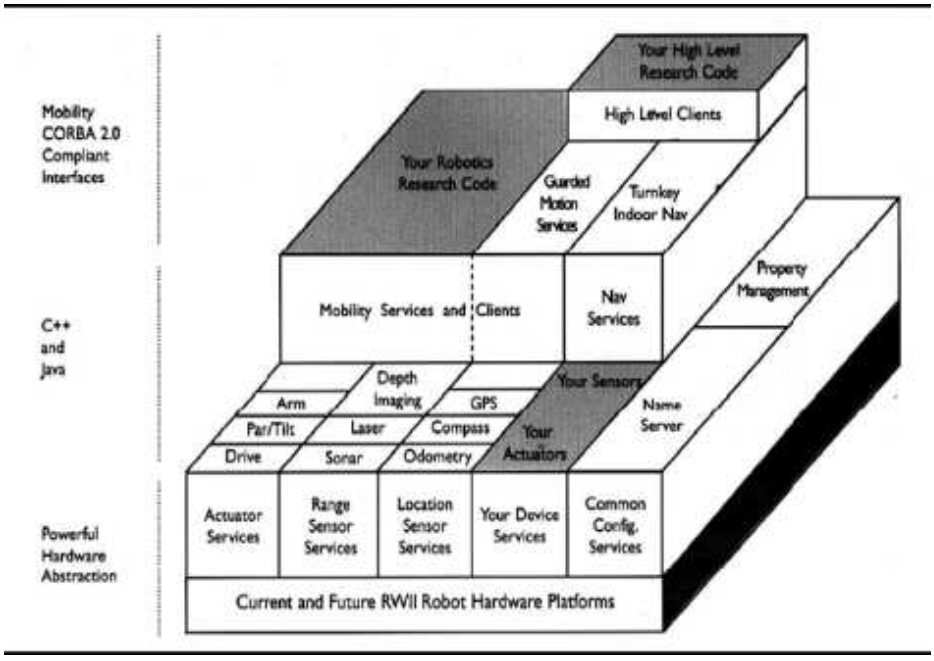

Figura 5.2: L'ambiente Mobility

getti distribuiti e organizzati gerarchicamente. Ogni oggetto è una unità software separata con una identità, delle interfacce e uno stato. Gli oggetti rappresentano le astrazioni del robot, dei sensori, degli attuatori, dei processi percettivi e della memoria. L'uso degli oggetti fornisce un modello flessibile di un sistema robotico, che può essere riconfigurato con nuovo hardware, nuovi algoritmi e nuove applicazioni. Il *Mobility Class Framework* (MCF) è un insieme di classi specifiche per il linguaggio, le quali forniscono l'implementazione per le funzionalità del sistema. Queste classi implementano le interfacce necessarie per l'integrazione con il software Mobility.[iRo02]

5.1.1 Una panoramica sul Mobility

Dato che tutti gli oggetti nell'ambiente Mobility condividono un insieme di interfacce, essi sono chiamati *System Components*. Alcuni di essi supportano altre interfacce che permettono loro di contenere altri oggetti, eseguirli come thread, o far fare loro altre funzioni. In Mobility ogni oggetto è sia cliente rispetto ad altri oggetti che servente per altri ancora. Per questo i termini cliente e servente vengono applicati soltanto durante l'invocazione di uno specifico metodo. L'oggetto che invoca il metodo è il cliente del metodo e l'oggetto che lo sta eseguendo è il servente. La definizione delle interfacce è scritta usando lo standard CORBA, in modo da ottenere maggiore indipendenza da particolari piattaforme o linguaggi di programmazione.

Mobility utilizza CORBA per comunicare tra gli oggetti tramite la libreria condivisa chiamata *Object Request Broker* (ORB). Sostanzialmente essa è una libreria di comunicazione che permette l'accesso trasparente ad oggetti in diversi spazi di indirizzamento sullo stesso o tra differenti computer.

5.1.2 Il Modello a oggetti

Il Modello a Oggetti del Robot Mobility (MROM) definisce le interfacce e gli oggetti necessari a rappresentare e gestire il software del robot come un insieme di esecuzioni concorrenti avente componenti software distribuite. Queste componenti rappresentano le astrazioni software dell'hardware e dell'ambiente del robot. MROM è definito usando un approccio Object-Oriented che è basato su CORBA 2.x. Le interfacce sono definite con le Interface Definition Language (IDL).

MROM descrive un robot come: " una collezione gerarchica di istanze di oggetti che forniscono interfacce ad ogni componente del sistema robotico."

In cima al MROM c'è il naming service standard del CORBA 2.x. Quest'ultimo, anche chiamato server dei nomi, permette l'accesso a molti elementi del sistema e contiene una directory di oggetti del robot e oggetti condivisi di supporto, compresi quelli per la gestione di multi robot.

Ogni robot ha il suo SystemModuleComponent, che fornisce un insieme di interfacce che permettono l'accesso a tutte le componenti di ciascun robot. Ogni SystemModuleComponent contiene un insieme di SystemComponents. Le SystemComponents rappresentano utili astrazioni di

- robot hardware;
- robot software;
- comportamenti;
- memoria;
- processi percettivi.

Tipici SystemComponents di un robot sono l'odometria, sensori tattili, sonar e attuatori.

5.1.3 Robot come gerarchia

Le componenti base del Mobility forniscono le astrazioni a basso livello dei sensori, attuatori e proprietà fisiche sotto forma di MobilitySystemComponents.

In Mobility, un robot è una collezione di elementi ordinata gerarchicamente. Sulla cima di questa gerarchia c'è il SystemModuleComponent, il quale contiene i sottosistemi separati di un robot individuale. Ognuno di questi sottosistemi è a sua volta un CompositeSystemComponent. I sottosistemi

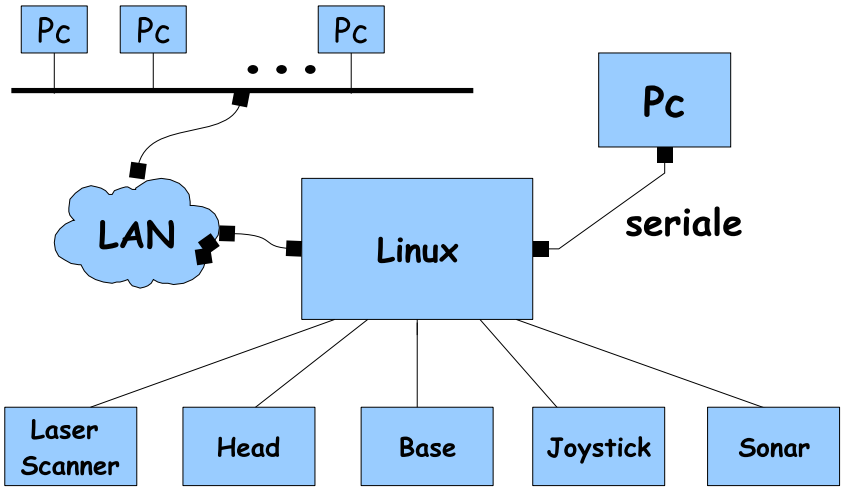

Figura 5.3: Componenti di sistema

contengono StateObject dinamici, aggiornati sullo stato dei sensori, degli attuatori e di tutte quelle proprietà sfruttabili da un qualche programma a runtime. Tramite le interfacce di questi oggetti, i programmi clienti possono ottenere:

- informazioni sullo stato del robot
- la posizione dei sensori del robot
- le proprietà dei sensori del robot
- la forma generale del corpo del robot
- altre informazioni geometriche di base determinate dall'hardware stesso

I programmi clienti comandano al robot di muoversi attraverso l'aggiornamento dello stato degli oggetti contenuti nel Drive Object. Gli oggetti di base, specialmente i sensori, forniscono diversi valori d'informazione sul loro

stato, sia valori sensoriali che informazioni geometriche grezze, oppure punti relativi al robot o punti di contatto.

Il concetto di stato è rappresentato dagli StateComponents. Questi oggetti forniscono semplici buffer di memoria che permettono di comunicare i cambiamenti di stato.

5.1.4 Mobility StateComponents

Gli StateComponents sono rappresentazioni dello stato aggiornate dinamicamente. Alcuni StateComponents sono anche oggetti contenitori e possono avere molte sotto-rappresentazioni dello stato, che di fatto sono considerate come diverse viste o diversi livelli di dettaglio. Gli StateComponents costituiscono uno "spazio percettivo". I cambiamenti sono comunicati tramite la propagazione di notifiche dagli stati ai clienti registrati. Si può pensare a uno StateComponent come ad una piccola lavagna su cui leggere e scrivere in tutta sicurezza. Le interfacce standard del Mobility permettono agli oggetti e alle varie rappresentazioni di essere estese nel tempo venendo in tal modo incontro ai nuovi sviluppi tecnologici.

Gli StateComponents che rappresentano il livello più basso dell'hardware robotico sono aggiornati dinamicamente e in maniera asincrona. Oggetti multipli del server interagiscono con le porte di I/O del robot e aggiornano il livello più basso degli StateComponents. Le notifiche di cambiamento da parte di questi aggiornamenti avviano il processo e le trasformazioni che portano alle astrazioni ad un più alto livello.

Tutti gli StateComponents hanno interfacce simili, ma esistono vari tipi di queste componenti per gestire varie caratteristiche del sistema. I principali StateComponents sono:

- BVectorState, vettori di byte, utili per dati sensoriali grezzi.

- FVectorState, vettori in virgola mobile, utile per la conversione dei dati sensoriali in metri.
- PointState, insieme di punti 3D spaziali, come i punti rilevati dai sonar.
- SegmentState, insieme di segmenti lineari 3D spaziali, come i raggi sonori.
- ActuatorState, rappresenta lo stato di un attuatore o il valore di un comando per un attuatore.

5.2 Sistemi sensoriali

I sistemi sensoriali di un robot sono rappresentati da un CompositeStateComponent, le cui proprietà descrivono le caratteristiche rilevanti del sistema sensoriale. Ad esempio, l'angolo di divergenza dei sonar è memorizzata come proprietà nel Sonar CompositeStateComponent. Possono avere un insieme di StateComponents considerati come oggetti figli del componente sensoriale. Questo insieme di figli rappresenta varie "viste" dello stato del sensore. Per i sonar queste viste includono una scala di valori grezzi (rappresentati da un componente FVectorState) oppure punti (componenti PointState) o segmenti lineari (componente SegmentState). La combinazione di diverse viste dello stato e di proprietà del sistema sensoriale permette un più facile adattamento alle variazioni geometriche nel sistemi robotici.

5.2.1 Sonar

L'oggetto sonar è un CompositeSystemComponent e contiene diversi StateObjects che rappresentano lo stato dei sensori del robot. L'oggetto sonar contiene un FVectorState che rappresenta la lettura dei valori grezzi del sonar

da parte del robot. Questa informazione è pensata principalmente per scopi di debugging e di test, dato che i sensori offrono le letture anche sotto forma di oggetti PointState, che restituiscono dei punti spaziali per ogni sonar in coordinate relative al robot.

Un oggetto SegmentState fornisce segmenti di linee per ogni lettura di un sonar, questa linea spaziale inizia dall'origine dell'impulso sonoro fino al suo punto d'arrivo.

5.2.2 Come lavorano i Sonar

Il SONAR, che deriva dall'acronimo SOund Navigation And Ranging, modella i contorni di un ambiente tramite la propagazione di onde sonore in accordo a come queste vengono catturate e riflesse. La sorgente genera un onda sonora, la quale, viaggiando, forma un cono che mano a mano si espande, mentre il sensore resta in ascolto di un ritorno dell'onda. Le caratteristiche di quest'eco possono aiutare a localizzare gli oggetti. I sonar sul robot dell'**iRobot** forniscono un'utile mappa dell'ambiente circostante al B21r, sempre che si rispettino gli intervalli di lettura previsti dalla **iRobot**. Per ogni lettura, il tempo totale tra la generazione dell'impulso sonoro e la ricezione del suo eco, insieme alla velocità di propagazione del suono nell'ambiente, genera una stima della distanza dell'oggetto che ha fatto rimbalzare l'eco. In contemporanea all'invio delle onde sonore e alla ricezione degli echi vengono aggiornate le rispettive strutture dati. Ogni sensore può individuare gli ostacoli che si trovano nel raggio d'azione a forma di cono, che iniziando dal sensore del robot con un angolo di circa 15 gradi si propaga verso l'esterno. Le caratteristiche della superficie di un oggetto, come quella di essere ruvido o levigato, così come pure l'angolo in cui l'oggetto è piazzato rispetto al robot, influiscono significativamente su come tale oggetto viene percepito. Per questo

bisogna tener conto che i dati provenienti dai sonar non sono infallibili e sarebbe opportuno effettuare letture multiple e controlli incrociati.

I sonar possono essere ingannati per le seguenti ragioni

- i sensori non hanno modo di conoscere esattamente dove, nei loro 15 gradi e nell'ampiezza del cono, sia realmente un ostacolo.
- i sensori non hanno modo di sapere l'angolo relativo di un ostacolo. Gli ostacoli con forti angoli acuti, potrebbero far rimbalzare le onde sonore in una direzione completamente differente. In questo modo i sensori non capteranno alcun ostacolo, almeno finché non riceveranno un qualche eco.
- se il segnale rimbalza su un oggetto fortemente obliquo su di un altro oggetto dell'ambiente e, successivamente, l'eco prodotto torna al sensore, allora quest'ultimo può sovrastimare la distanza tra il robot e l'ostacolo più vicino. Questo effetto è chiamato riflessione speculare.
- muri estremamente lisci che presentano angoli molto ripidi, e muri a vetro, possono seriamente trarre in inganno i sensori.

Per cercare di porre rimedio a tutti questi problemi il B21r è stato dotato di molteplici sensori sonar, che offrono ridondanza e permettono il controllo incrociato. I sonar quasi mai sottostimano la distanza di un ostacolo. Comunque, è una buona idea esaminare le distanze ottenute da un insieme di sonar e usare soltanto i valori più bassi. Oppure, si possono memorizzare letture multiple mentre il robot è in movimento, e usare i dati accumulati per costruire una griglia d'occupazione (occupancy grid). Se diverse letture, da diversi angoli e da diversi sensori, individuano un ostacolo pressappoco nello stesso posto, molto probabilmente è sicuro marcare quella zona come occupata.

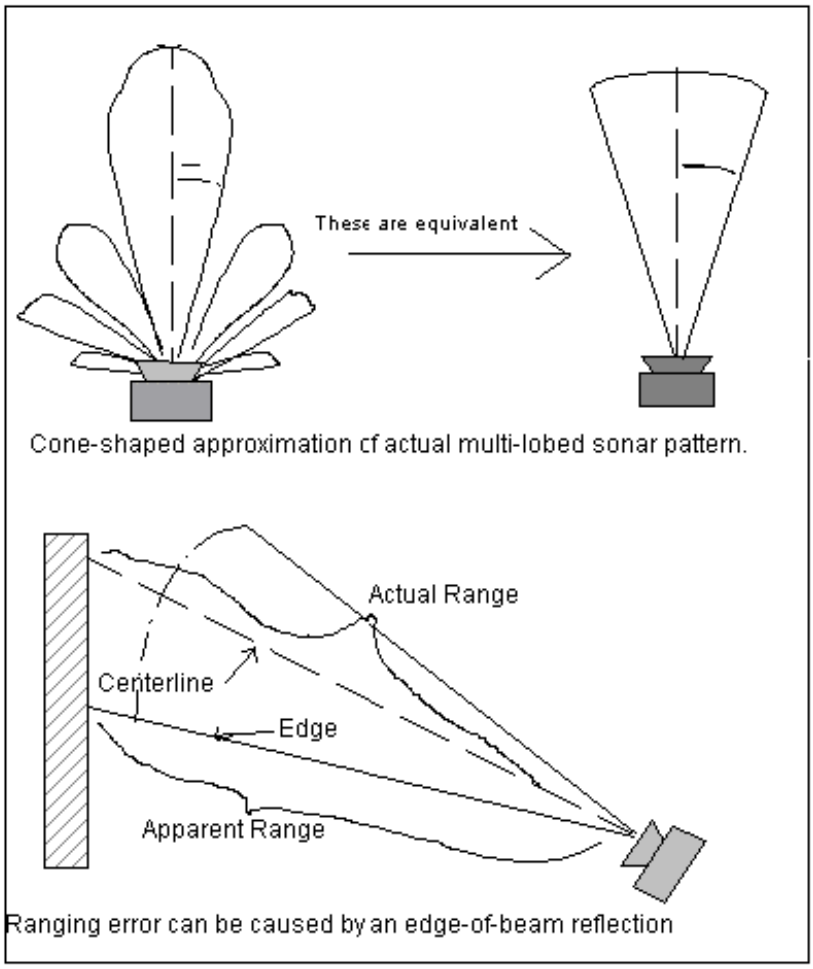

Figura 5.4: Errori di allineamento

5.2.3 Come i sonar possono essere ingannati

Quando il raggio sonoro del sonar colpisce una superficie con un angolo di incidenza molto largo, l'onda non è riflessa sulla linea centrale, ma trasversalmente. In più, a causa della larghezza relativa del raggio, all'incirca un angolo di 15 gradi, il sensore tende a produrre una immagine piuttosto confusa dell'ambiente. Questo può portare a un errore angolare, che influisce sulle risposte sensoriali del robot in maniera simile all'errore radiale.

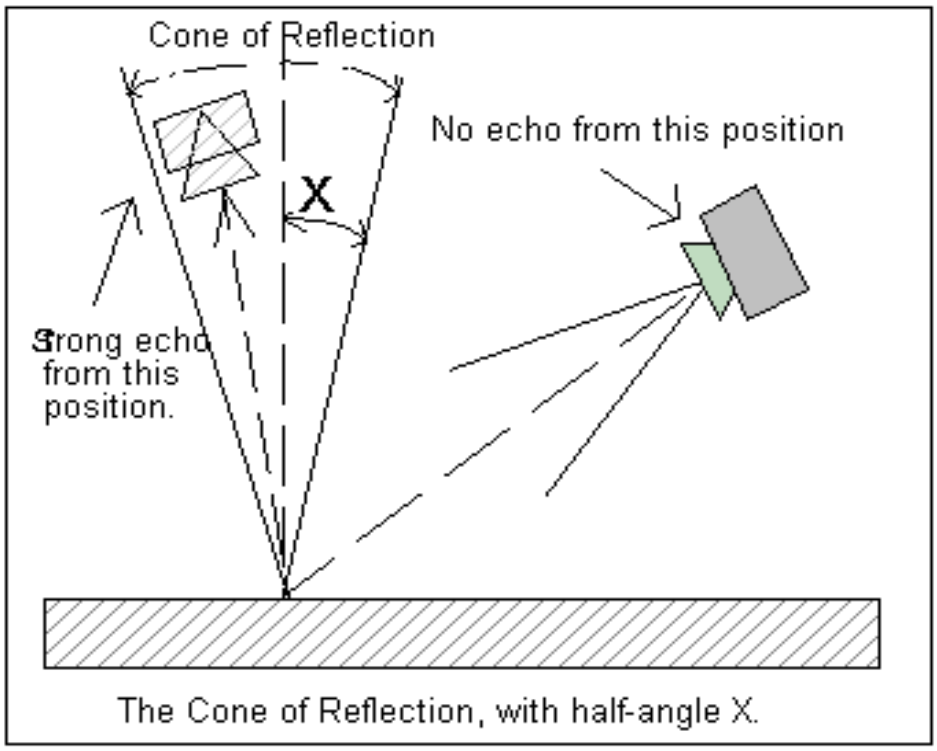

Figura 5.5: Errori angolari

Dopo aver colpito una superficie con un ampio angolo di incidenza, l'eco può andar perduto oppure tornare indietro sul ricevitore del sonar con più forza. Questo tipo di falsa riflessione accade quando l'angolo di incidenza del raggio sonoro è più grande di un angolo limite, indicato nella Figura 5.5 con X, che definisce il cono di riflessione (CR) per la superficie. Un raggio sonoro colpisce un muro e viene prodotto un cono di riflessione lontano dal sensore, creando un lungo raggio sonoro e dando così l'impressione di non avere ostacoli. Apparentemente il raggio sonoro penetra il muro. Ogni materiale ha il suo angolo CR, che va dai 7 o 8 gradi per il vetro, ai quasi 90 gradi per superfici ruvide.

5.2.4 L'odometria: l'oggetto RobotDrive

L'oggetto RobotDrive è un CompositeSystemComponent. Contiene due diversi StateObject, più precisamente ActuatorState, che rappresentano lo stato del sistema di guida del robot e il comando corrente di guida.

L'oggetto Odometry è un CompositeSystemComponent. Contiene diversi StateObject che rappresentano lo stato dell'odometria del robot. Un oggetto TransformState rappresenta la locazione corrente del robot. Un oggetto FVectorState rappresenta la velocità corrente del robot.

La base mobile del robot è equipaggiata con degli encoder che tengono traccia delle rotazioni delle ruote mentre il robot si muove nell'ambiente. Il controllore di movimento del robot integra queste misure per cercare di stimare la sua posizione corrente in ogni istante rispetto alla sua posizione di partenza, che è considerata quella in cui ha iniziato a muoversi. Queste misurazioni sono molto accurate per le piccole distanze, ma errori si accumulano mano a mano che il robot si muove per lunghi percorsi. Da solo, il controllore hardware del movimento del robot, non ha modo di individuare gli slittamenti delle ruote o eventuali errori nel tracking delle ruote. Ambienti con pavimenti scivolosi, con tappeti o stipiti rendono questo lavoro ancora più difficile.

Mobility fornisce informazioni sull'odometria relative alla posizione di partenza del robot, sotto forma di valori numerici sulle X, per i movimenti in avanti e indietro, sulle Y, per i movimenti laterali e un angolo theta relativo alle rotazioni. Allo stesso modo si può ottenere la velocità. L'output è in metri e radianti e la velocità in metri al secondo e radianti al secondo. Con Mobility, non è necessario avere a che fare con i dati grezzi degli encoder. Mobility aggiorna l'odometria molto rapidamente ($>$ 10Hz).

5.2.5 Come vengono processati i dati degli encoder

Il controller rFlex del robot maschera i dati grezzi ottenuti dagli encoder e combina i dati di tutti gli encoder in due assi virtuali, l'asse di traslazione e quelli di rotazione. Questi "encoder virtuali" sono inviati al software Mobility. Questo stesso invia comandi di rotazione e traslazione a questi assi virtuali. Il server di base del Mobility inoltre converte questi interi in unità effettive (metri e radianti). La velocità degli assi è essa stessa riportata e convertita in metri al secondo e radianti al secondo. Mobility ottiene aggiornamenti dagli assi virtuali in circa 10-15Hz. La combinazione degli encoder in assi virtuali corregge lo slittamento delle ruote in rotazione e quando attraversano terreni accidentati forniscono la migliore valutazione dei movimenti di traslazione e rotazione (posizione e velocità).

iRobot non permette in alcun modo l'accesso agli encoder per non compromettere la sicurezza del sistema di controllo del robot.

5.3 Panoramica del CORBA

CORBA (Common Object Request Broker Architecture) è lo standard per architetture ad oggetti distribuite sviluppato dal consorzio dell'Object Management Group (OMG). Si tratta di un'architettura per la realizzazione di un open software bus, appunto l'Object Request Broker (ORB). Utilizzando questo bus, applicazioni ad oggetti eterogenee possono interoperare attraverso la rete, indipendentemente dall'architettura e dai sistemi operativi, e soprattutto dal linguaggio di programmazione. Le applicazioni (ed in particolare i componenti ad oggetti utilizzati da queste) possono essere scritti in linguaggi differenti. In questo modo un programma scritto in C++ può

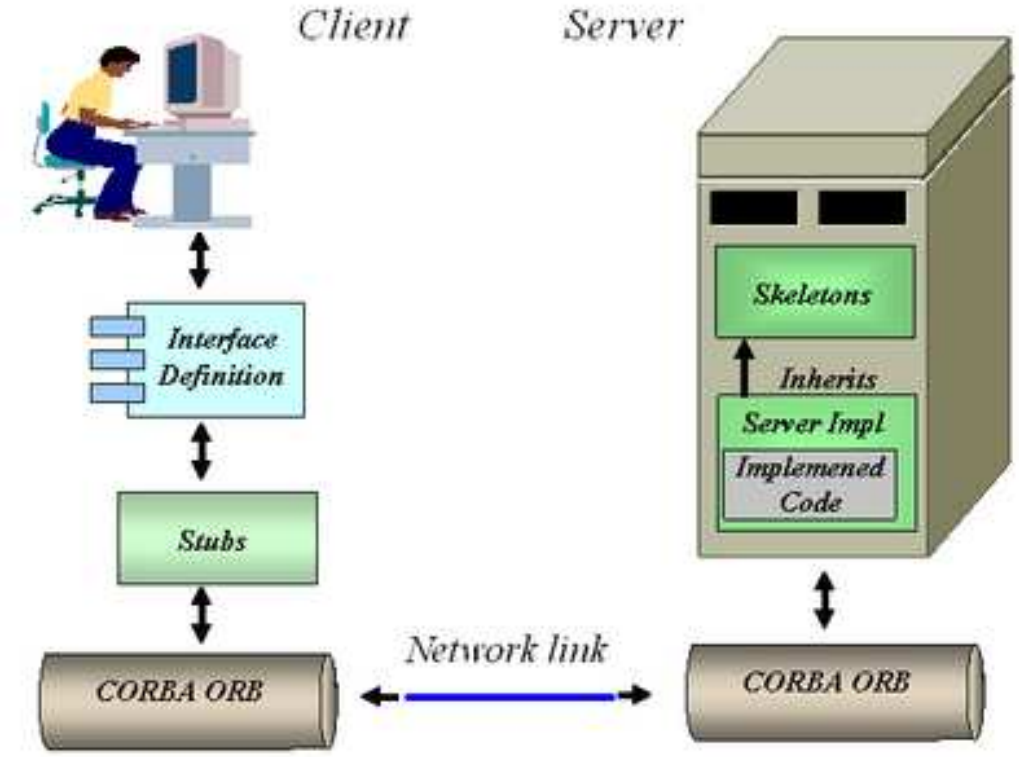

Figura 5.6: Chiamata di un metodo di un oggetto remoto

richiamare i metodi di un oggetto (magari remoto) scritto in Java, oppure in Smalltalk, o addirittura in COBOL.

Ovviamente oggetti scritti in linguaggi differenti, in esecuzione su Sistemi Operativi differenti ed architetture completamente diverse non possono comunicare direttamente, ma devono utilizzare certe interfacce (e protocolli) e comunicare attraverso dei proxy che comunque rendono totalmente trasparenti i dettagli della comunicazione: una volta ottenuto un riferimento ad un oggetto remoto si potrà richiamare semplicemente un suo metodo nel modo usuale.

Per far questo si ha uno stub (dalla parte del client) ed uno skeleton (dalla parte del server) che fungono appunto da proxy.

Un servant è un'istanza dell'implementazione di un oggetto CORBA, mentre un server è un processo che istanzia i servant e li mette a disposizione dei client, che sono applicazioni che invocano metodi di oggetti CORBA. Un client di un oggetto CORBA richiama i metodi di un oggetto remoto tramite un object reference che ottiene dal server. Lo stub, tramite l'ORB, identifica la macchina su cui è presente il server CORBA che gestisce l'oggetto del quale si vuole richiamare il metodo. Una volta stabilita la connessione con tale macchina, la richiesta, insieme ad i vari eventuali parametri con cui si richiama il metodo, vengono spediti dall'ORB del client all'ORB del server; quest'ultimo ORB presenta la richiesta allo skeleton che provvede a richiamare il metodo del servant, e, una volta terminato, a consegnare i risultati al client (ovviamente sempre tramite l'ORB). Quindi, come già detto, tutte le comunicazioni sono trasparenti al client, che penserà di invocare semplicemente un metodo di un oggetto normale. I due ORB non devono essere necessariamente istanze della stessa implementazione: basta che implementino gli standard di CORBA; del resto questo è tipico dei protocolli di Internet: quando si accede ad un sito FTP, vi si accede indipendentemente dalla particolare implementazione del server, in quanto viene utilizzato un protocollo standard.

I dati prima di essere spediti vengono memorizzati in un formato opportuno standard di CORBA; tale processo viene detto marshaling. Il recupero dei dati sul server viene detto a sua volta unmarshaling.

Ovviamente, poiché tali oggetti possono essere implementati con linguaggi di programmazione differente, ci deve essere un modo tramite il quale la comunicazione possa avvenire; del resto per chiamare i metodi di un oggetto non se ne deve conoscere l'implementazione, ma solo conoscerne l'interfaccia (questo è vero nella normale programmazione, ed ancora di più nella program-

mazione ad oggetti che esalta il concetto di information hiding). Quindi è sufficiente avere a disposizione un modo standard per definire le interfacce degli oggetti CORBA.

Il metodo per definire tali interfacce è, tanto per cambiare, un linguaggio di programmazione, che fa parte dello standard CORBA: questo linguaggio si chiama IDL, ovverosia Interface Definition Language. Oltre all'ORB, quindi è necessario un compilatore per le IDL.

Il linguaggio di programmazione in cui è stato svolto il presente lavoro di tesi è *Java Standard Edition 6*. Pertanto in questo caso si è fatto uso dello specifico compilatore messo a disposizione dalla Sun, *idltojava*. [BVD01]

Al compilatore si dà un programma scritto appunto in IDL, che definisce solamente l'interfaccia dell'oggetto CORBA, quindi il linguaggio IDL è effettivamente un linguaggio puramente dichiarativo. Il compilatore produrrà in output dei file che descrivono questa interfaccia nel linguaggio appropriato; in questo caso si tratterà di un'interfaccia in Java, ma se si dà in input lo stesso file ad un compilatore IDL per linguaggio C++ si otterrà in output un programma con una (o più) classe in C++ (probabilmente si tratterà di una classe astratta, che può essere intesa come un'interfaccia in C++).

In particolare il compilatore genererà anche i file per lo stub e per lo skeleton. Poiché al compilatore viene fornita solo l'interfaccia, le classi generate dovranno essere utilizzate (tramite la derivazione o l'implementazione di un'interfaccia) per l'effettiva implementazione, che spetta sempre al programmatore sia del server che del client.

Il client recupererà un riferimento all'oggetto CORBA tramite l'ORB locale; una volta che si è ottenuto tale riferimento lo si deve convertire nel tipo giusto (operazione detta *narrowing*, in quanto si deve far scendere l'oggetto nella gerarchia, fino al punto giusto). Si ottiene infatti, come object reference,

un org.omg.CORBA.Object. Il casting deve essere effettuato esplicitamente perché il runtime di Java non può sempre sapere il tipo esatto di un oggetto CORBA. Ovviamente si utilizzeranno metodi di classi di supporto (dette appunto helper), che vengono create appositamente dal compilatore.

Tramite il naming service si recupererà un riferimento all'oggetto remoto, specificandone il nome. Per ottenere un object reference al name server si utilizza il metodo resolve passandogli la stringa "NameService" che è definita per tutti gli ORB. Il narrowing (cioè il cast) è effettuato richiamando il metodo narrow della classe NamingContextHelper, fornita da CORBA. A questo punto si ha effettivamente il NamingContext che si utilizzerà per ottenere un riferimento ad un oggetto; per fare questo il client deve specificare un array di oggetti NameComponent che rappresenta il path dell'oggetto sul server (che quindi deve essere conosciuto dal client). Dopo che il riferimento è stato sottoposto a narrowing utilizzando l'helper creato dal compilatore, si ha effettivamente un oggetto del tipo cercato, e si possono richiamare i metodi di tale oggetto remoto.

Capitolo 6

Prove sperimentali e discussione dei risultati

Le prove delle applicazioni MapBuilding e Navigation si sono svolte usando come robot il B21r descritto nel capitolo precedente. La sequenza delle prove è stata la seguente:

- Costruzione della mappa con l'applicazione MapBuilding
- Prova di navigazione con Navigation senza Predizione
- Prova di navigazione con Navigation con predizione

Nei prossimo paragrafi vedremo nel dettaglio queste fasi.

6.1 Costruzione della mappa con l'applicazione MapBuilding

Le prove del MapBuilding sono state eseguite settando i seguenti parametri:

- griglia per la mappa con dimensione di un quadrato di 50x50 celle;

- velocità del robot 10 cm/sec;
- dimensione delle celle impostata a 20 cm^2; tale scelta è legata alla velocità del robot. Per ogni cella attraversata vengono effettuate 4 letture, ciò comporta una buona certezza nella localizzazione;
- letture dal robot ogni 0.5 sec; i dati dei sonar vengono aggiornati dal software del robot ogni 0.3 sec, in questa maniera si ha la certezza di elaborare ogni volta una lettura diversa;
- velocità di controllo di nuove istruzioni da parte del Thread dei comandi settata a 0.1 sec;
- frequenza aggiornamenti da parte dell'algoritmo HIMM ogni 0.25 sec;
- suddivisione dei 24 sensori del robot in gruppi di 3, in modo tale da avere una media dei valori lungo le otto direzioni cardinali, vedere 4.1.1;
- i valori di aggiornamenti dell'HIMM impostati a -1 in caso di cella libera e +5; i test effettuati con un valore di incremento pari a +3 dell'implementazione standard dell'HIMM, hanno portato a risultati mediocri con valori di certezza degli ostacoli molto bassi. Incrementando tale variabile si è ottenuta una migliore precisione della mappa.

L'ambiente su cui abbiamo effettuato tutte le prove è visibile in figura 6.1. E' stato costruito "ad hoc" in modo da avere un ostacolo con una certa complessità. Si è voluto verificare, in tal modo, la capacità dell'algoritmo di individuare in maniera piuttosto precisa la forma geometrica propria dall'ostacolo. In fig. 6.2 è mostrato il risultato finale dopo la normalizzazione. Come si può vedere, la figura al centro è stata modellata in maniera non perfetta, ma sufficiente a dare l'idea della forma dell'ostacolo. Alla sinistra

Figura 6.1: Ambiente in cui sono state effettuate le prove

dell'ostacolo, c'è una singola cella, isolata e marcata come occupata, è evidente che ciò dipende da un errore di lettura da parte dei sensori. Alla destra, è stata modellato il lato di una panchina in pietra e ferro, a dimostrazione del fatto che questi due materiali vengono ben individuati dai sonar.

6.2 Prova di navigazione con Navigation senza Predizione

È stata eseguita questa prova per verificare il funzionamento corretto degli algoritmi di navigazione classici e valutare le loro prestazioni in modo da

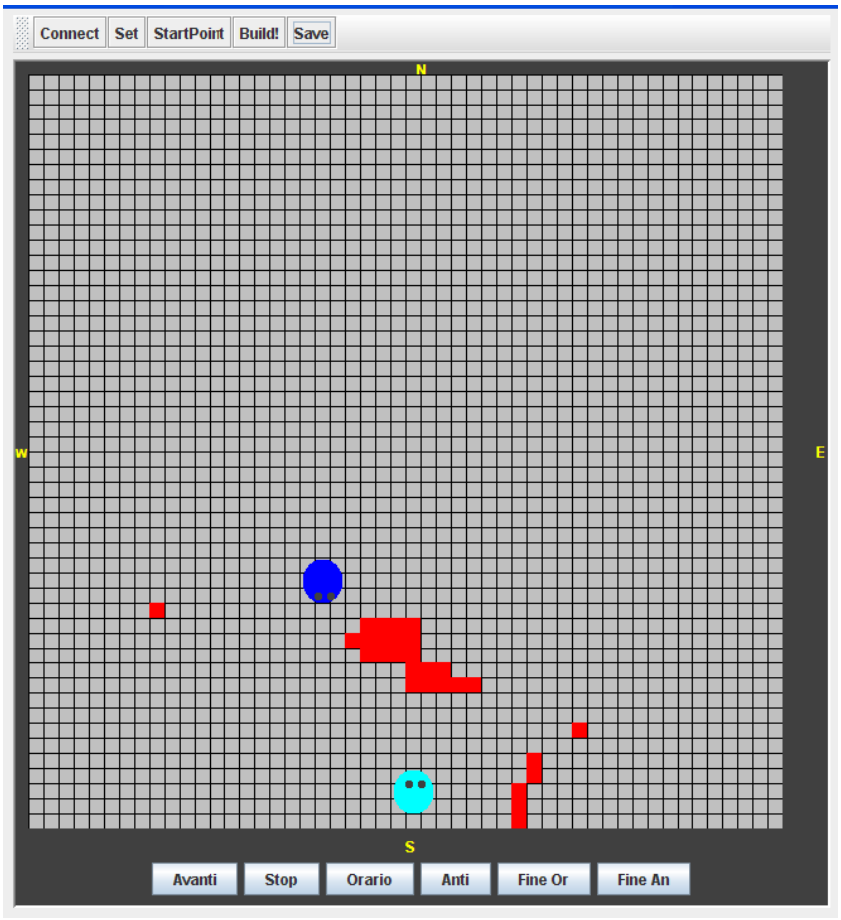

Figura 6.2: Risultato del MapBuilding dopo la normalizzazione

poterle poi confrontare con la navigazione con predizione. Questa prova è stata eseguita settando i seguenti parametri:

- griglia creata dall'applicazione MapBuilding e caricata dal database;
- la velocita del robot impostata a 0,1 m/sec;
- tempo di controllo della localizzazione e individuazione del nuovo sottoobiettivo, impostato a 2 sec;
- distanza di sicurezza per individuazione dell'ostacolo a 1m;

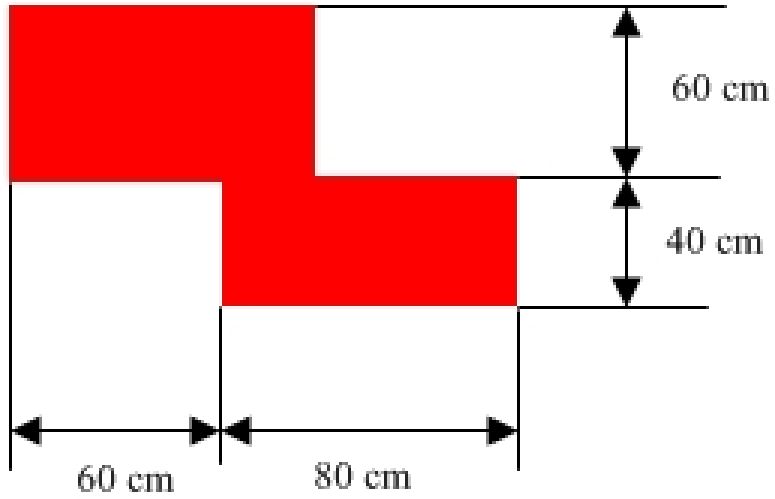

Figura 6.3: Vista schematica dell'ambiente dall'alto

- 5 sec, tempo d'attesa per stabilire se un oggetto è in movimento.

Durante la prova, il robot si è fermato più volte di fronte agli ostacoli per controllare se questi risultassero già modellati nella mappa. Questa operazione è necessaria per valutare un'eventuale modifica del percorso e i comandi motori devono essere interrotti per evitare che il robot andasse ad urtare contro l'ostacolo. Molti degli ostacoli incontrati risultavano ininfluenti ad una modifica del path che si stava seguendo, ma il sistema non aveva modo di saperlo almeno finché non avesse controllato la posizione dell'oggetto sulla mappa. Questo ha portato a un rallentamento nel raggiungimento dell'obiettivo.

La localizzazione è risultata esatta, il tragitto creato ad ogni prova è risultato essere il più breve per arrivare a destinazione e gli ostacoli venivano individuati e, se necessario, modellati ed evitati.

In definitiva, il sistema è risultato funzionante, ma il costo computazionale per il controllo dei dati sulla mappa è troppo elevato per potersi considerare utilizzabile in un'applicazione che deve funzionare in tempo reale.

6.3 Prove di Navigation con Predizione

Questa prova è stata eseguita settando i seguenti parametri:

- La velocita del robot impostata a 0,1 m/sec;
- L'intervallo di ogni predizione è di 0.3 sec;
- Predizione degli dati ad una distanza massima di 1m;
- L'offset da aggiungere alla misura dei sensori per confrontarla con la distanza predetta è di 30 cm;
- 5 sec, tempo d'attesa per stabilire se un oggetto è in movimento.

Durante questa prova il robot ha eseguito il suo cammino in maniera fluida; come si può vedere in figura 6.4 step1, si è avvicinato agli ostacoli che si trovavano nel mezzo tra il suo punto di partenza ed il goal, ha ruotato in senso antiorario di 90° ed ha proseguito nel suo cammino fino allo step 3. In quel momento si è accorto di avere un ostacolo non modellato nella mappa: in realtà era la mappa ad essere imprecisa. L'errore deriva dal processo di creazione della mappa, dovuto a errori di lettura dei sonar in aggiunta ai vari processi di approssimazione dell'algoritmo e all'accumulo di errori odometrici. Nonostante questa situazione, il robot ha modellato sulla mappa il nuovo ostacolo, l'ha evitato e nello step 5 è arrivato a destinazione.

Il tempo per giungere all'obiettivo è stato di 1'20" circa. Durante il percorso il robot non ha avuto problemi e alla velocità di 0.1 m/sec risulta un tempo di percorrenza adeguato.

Aumentando la velocità, si sono riscontrati problemi di ritardo di comunicazioni nella rete wireless. Il robot avanzava troppo velocemente nella mappa rispetto al controllo sulla localizzazione, oppure non recepiva tempestivamente un corretto comando di movimento. Questa tipologia di problemi può essere risolta collegando via cavo il computer remoto o implementando il sistema in locale.

Durante le prove effettuate, abbiamo riscontrato un corretto comportamento dei sonar nel rilevare la presenza di persone e oggetti metallici. Le vibrazioni dovute alla pavimentazione non perfettamente liscia hanno comportato qualche errore nei dati odometrici, errori peraltro poco influenti nel nostro caso, in quanto la distanza percorsa non è stata eccessiva.

6.4 Discussione dei risultati

La differenza principale nei due sistemi di navigazione si ha durante la rilevazione degli ostacoli. Il sistema che non fa uso di predizione deve trasformare i dati dei sonar in coordinate relative alla mappa, per controllare se l'ostacolo individuato è d'intralcio al suo percorso verso l'obiettivo. Il sistema con EP esegue l'operazione contraria, cioè dalle informazioni presenti nella mappa calcola i valori che i sonar dovranno restituire nell'istante successivo e, in seguito, effettuare il confronto.

Riassumendo, mentre il sistema senza EP è impegnato nella conversione dei dati in coordinate mappa e nell'accesso di quest'ultima, il sistema con EP deve effettuare soltanto una differenza tra valori in virgola mobile e

controllarne il risultato.

Come risultato si ha che, nel complesso, il sistema di navigazione che adotta la predizione è più veloce, anche se di poco, nell'interpretare le informazioni sensoriali e nel decidere quindi il path da seguire.

Si sarebbe avuta una differenza maggiore tra i due sistemi, a livello di prestazioni, se i dati sensoriali fossero stati più complessi da elaborare, in quanto sarebbe aumentato il tempo necessario al sistema senza EP per processare le informazioni.

Ad esempio, l'uso di telecamere avrebbe comportato una elaborata fase di analisi delle immagini per l'estrazione di informazioni utili. Il vantaggio nell'utilizzo della predizione sarebbe stato quindi più evidente, in quanto avrebbe permesso di anticipare i costosi calcoli di processing dei dati, basti pensare all'object recognition e al calcolo della distanza di oggetti tramite la visione stereoscopica.

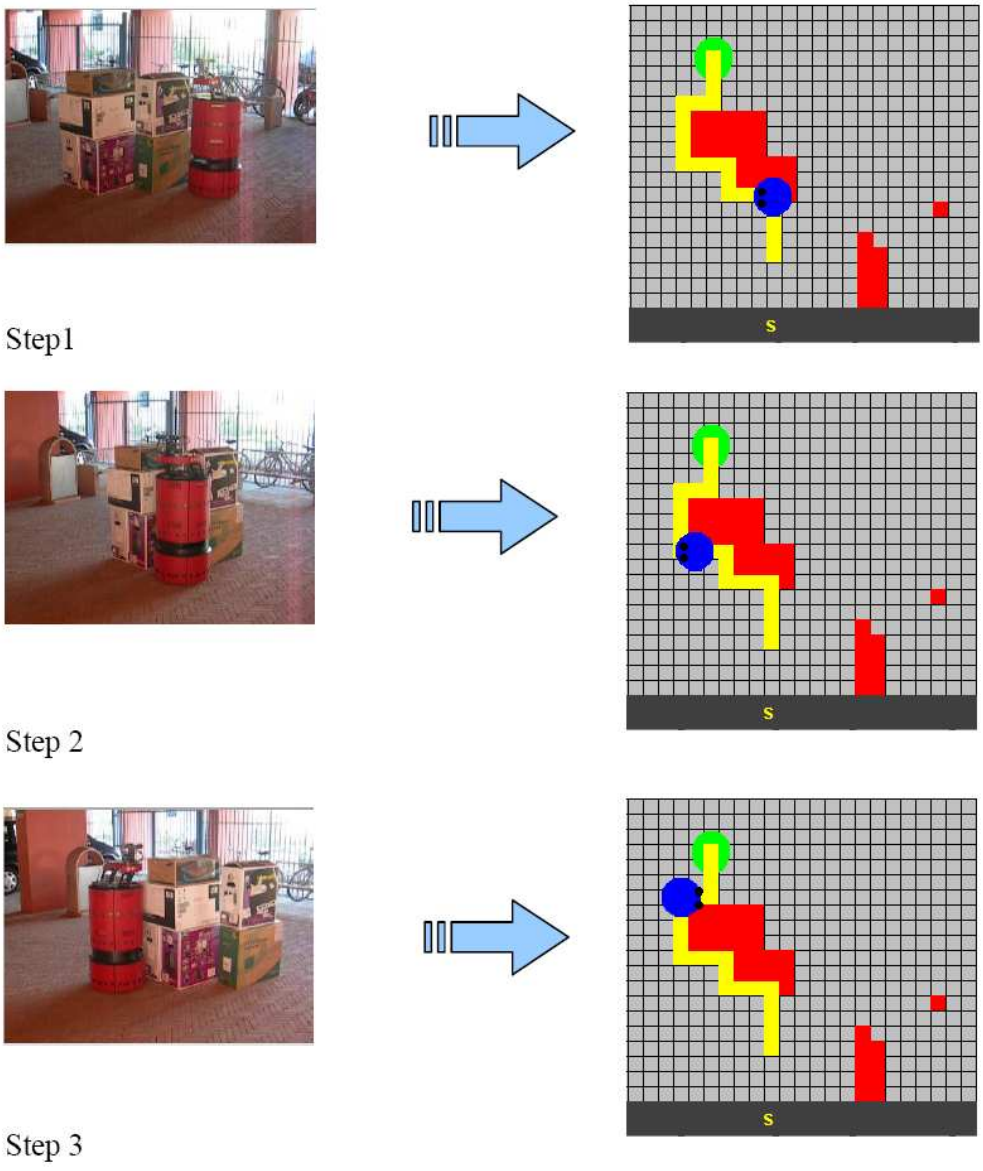

Figura 6.4: Applicazione Navigation con Predizione

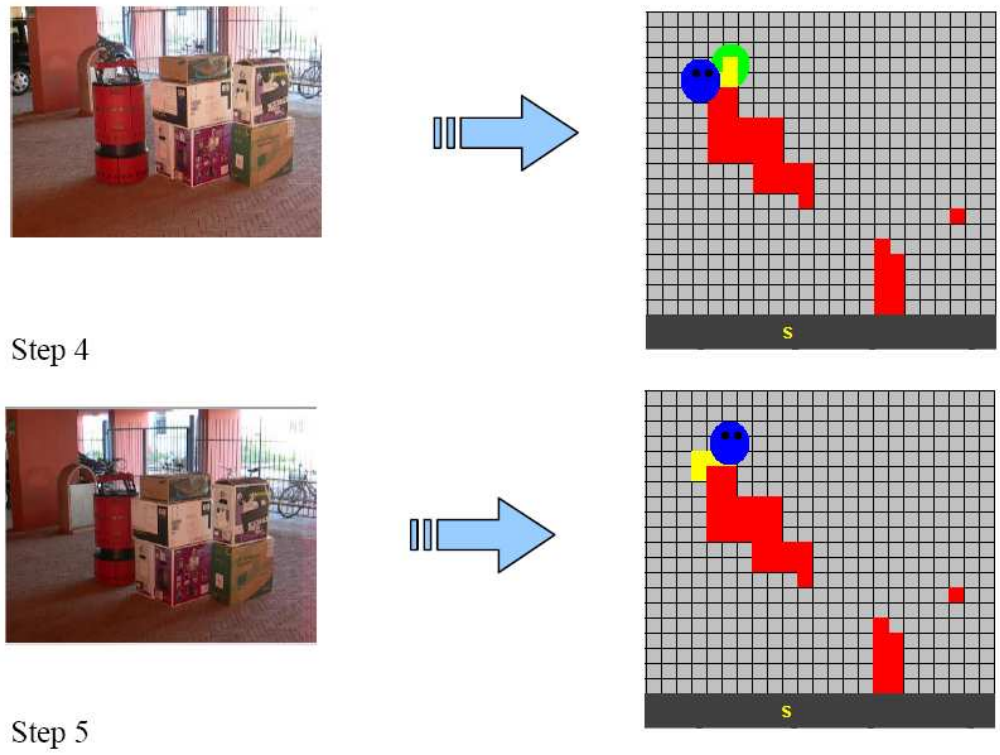

Figura 6.5: Applicazione Navigation con Predizione

Capitolo 7

Conclusioni

Nella nostra tesi abbiamo realizzato un sistema di navigazione autonomo per un robot mobile, basato sull'anticipazione sensoriale. Nella realizzazione di questo sistema, è stata sviluppata un'applicazione per la costruzione di una mappa per rendere l'ambiente solo parzialmente sconosciuto al sistema di navigazione e quindi per costruire quei modelli interni che permettono al robot di eseguire predizioni sensoriali. Questa applicazione (MapBuilding) fa uso delle griglie regolari per rappresentare il mondo e dell'algoritmo HIMM per la realizzazione del suo modello. MapBuilding utilizza i dati provenienti dai sonar del robot per avere una descrizione locale dell'ambiente. La scelta dei sonar è dovuta al fatto che essi costituiscono la tipologia di sensori più economica e semplice e più usata sui robot mobili; essi presentano però tutta una serie di problemi dovuti alle condizioni ambientali, come umidità, pressione, vento e alle superfici e ai materiali di cui sono composti gli oggetti da rilevare. Per ottenere dei risultati soddisfacenti, si deve far uso di particolari tecniche algoritmiche per l'approssimazione dei dati. Nonostante questo, siamo riusciti ad ottenere una mappa sufficientemente precisa dell'ambiente che ci ha permesso di poter testare il sistema di navigazione.

Il sistema di navigazione autonomo vero e proprio è stato implementato tramite l'applicazione Navigation. Tale applicazione è stata realizzata in due versioni, una con anticipazione sensoriale (EP: Expected Perception) e l'altra senza, per poterle poi confrontare. L'implementazione senza EP in alcuni casi ha presentato un problema: nell'identificare un nuovo ostacolo, il robot risultava lento nel calcolo, rischiando, a volte, di urtare contro l'ostacolo stesso. Del calcolo fanno parte le operazioni di localizzazione del robot all'interno della mappa, la traduzione dei dati sensoriali in dati compatibili con quelli della mappa e il confronto tra questi e i dati del modello interno. La soluzione trovata è stata dunque quella di bloccare il movimento del robot finché il calcolo non fosse terminato, causando una serie di fermate e ripartenze che rallentavano il raggiungimento dell'obiettivo. L'implementazione con EP non ha invece presentato queste problematiche. La predizione dei valori sensoriali, ha permesso di eseguire la traduzione dei dati della mappa in anticipo rispetto all'arrivo delle risposte sensoriali. In tal modo l'unica operazione da eseguire è il confronto tra numeri in virgola mobile. E' risultato quindi che il robot è riuscito a raggiungere l'obiettivo senza urti, manifestando maggiore velocità nell'interpretare le informazioni sensoriali. Da questo abbiamo dedotto che l'algoritmo con predizione ha una maggiore reattività ai cambiamenti dell'ambiente permettendo anche l'aumento della velocità di base del robot; questo rende più veloce il raggiungimento degli obiettivi.

Per quanto riguarda gli sviluppi futuri, sarebbe interessante aggiungere altri tipi si sensori, come le telecamere. Questo mostrerebbe ulteriori vantaggi nell'utilizzo dell'anticipazione sensoriale, in quanto i calcoli nell'elaborazione dei dati sensoriali risulterebbero più laboriosi e il sistema di navigazione senza EP andrebbe ancora di più in difficoltà. In più si avrebbe la possibilità di ottenere una rappresentazione del mondo più accurata, permettendo

una localizzazione e pianificazione più precisa. Inoltre si potrebbe applicare l'anticipazione sensoriale anche per la verifica della localizzazione eseguita tramite odometria. Anche in questo caso l'anticipazione sensoriale dovrebbe rendere il sistema più veloce e preciso.

Bibliografia

[AGO97] Albano, Ghelli, and Orsini. *Basi di dati relazionali e a oggetti.* 1997.

[Ark98] R. C. Arkin. *Behavior-based Robotics.* MIT Press, 1998.

[BEF96] J. Borenstein, H. R. Everett, and L. Feng. *Where am I?Sensors and Methods for Mobile Robot Positioning.* 1996.

[Bek05] G.A. Bekey. *Autonomous Robots: From Biological Inspiration To Implementation And Control.* 2005.

[Ber98] A. Berthoz. *Il senso del movimento.* 1998.

[BK91] J. Borenstein and Y. Koren. Histogramic In Motion Mapping for mobile robot obstacle avoidance, 1991.

[BMF] C. Buschmann, F. Muller, and S. Fischer. Grid-Based Navigation for Autonomous Mobile Robots.

[Bro86] R.A. Brooks. A robust layered system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1986.

[Bro90] R.A. Brooks. The behavior language, 1990.

[Bro91a] R.A. Brooks. Intelligence without representation, 1991.

[Bro91b] R.A. Brooks. New approaches to robotics, 1991.

[Bro99] R.A. Brooks. *Cambrian Intelligence*. 1999.

[BVD01] G. Brose, A. Vogel, and K. Duddy. *Java Programming with CORBA*. 2001.

[CB00] H. Choset and J. Burdick. Sensor Based Motion Planning: The Hierarchical Generalized VoronoiGraph. *International Journal of Robotics Research*, 2000.

[CC] A. Cappello and A. Cappozzo. *Bioingegneria della postura e del movimento*.

[CTP06] G. Calvi, A. Tutino, and G. Pezzulo. L'approccio anticipatorio in robotica: una analisi comparativa. In *Atti del Secondo Workshop Italiano di Vita Artificiale*, 2006.

[DFBT99] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte Carlo localization for mobile robots. volume 2, pages 1322–1328. IEEE Int. Conf. Robot. Autom. (ICRA), 1999.

[DG00] M. Desmurget and S. Grafton. Forward modeling allows feedback control for fast reaching movements. *Trends in Cognitive Sciences*, 4:423–431, 2000.

[Dij59] E. W. Dijkstra. A note on two problem in connexion with graphs. *Numerische Mathematik*, 1:269 – 271, 1959.

[DTL+03a] Datteri, Teti, Laschi, Tamburrini, Dario, and Guglielmelli. Expected perception: an anticipation-based perception-action scheme in robots. pages 934–939, Las Vegas, Nevada, 2003. IROS

2003, 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems.

[DTL+03b] Datteri, Teti, Laschi, Tamburrini, Dario, and Guglielmelli. Expected perception in robots: a biologically driven perception-action scheme. volume 3, pages 1405–1410. ICAR2003, 11th International Conference on Robotics and Automation, 2003.

[Fox98] Dieter Fox. *Markov Localization : A Probabilistic Framework for Mobile Robot Localization and Navigation.* PhD thesis, December 1998.

[GHJV02] Gamma, Helm, Johnson, and Vlissides. *Design patterns: elementi per il riuso di software a oggetti.* 2002.

[HM92] K. Hughes and R. Murphy. Ultrasonic robot localization using Dempster-Shafer theory. *SPIE Stochastic Methods in Signal Processing, Image Processing, and Computer Vision*, 1992.

[iRo02] iRobot. *Mobility Robot Integration Software Userś Guide*, 2002.

[Kaw99] M. Kawato. Internal models for motor control and trajectory planning, 1999.

[KG92] Kawato and Gomi. A computational model of four regions of the cerebellum based on feedback error learning. *Biological Cybernetics*, 69:95–103, 1992.

[KSJ00] E.R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science.* 2000.

[Lat91] J.C. Latombe. *Robot Motion Planning.* 1991.

[Mey90] A. Meystel. Knowledge based nested hierarchical control, 1990.

[Mur00] Robin R. Murphy. *Introduction to AI Robotics*. 2000.

[Neu] Neurobotics - The fusion of Neuroscience and Robotics. FP6-IST-001917.

[Per99] Andrea Peru. Basi neurofisiologiche della percezione, 1999.

[SS00] L. Sciavicco and B. Siciliano. *Robotica industriale*. 2000.

[Ste95] Anthony Stentz. The Focussed D* Algorithm for Real-Time Replanning, 1995.

[Ste99] Anthony Stentz. Optimal and Efficient Path Planning for Partially-Known Environment, 1999.

[Van06] Marco Vanneschi. Appunti di architetture parallele e distribuite, 2006.

[Vig06] Sebastiano Vigna. L'algoritmo di Dijkstra, 2006.

[WK98] Wolpert and Kawato. Multiple paired forward and inverse models for motor control., 1998.

[YSSB98] Yahja, Stentz, Singh, and Brumitt. Framed-Quadtree Path Planning for Mobile Robots Operating in Sparse Environments. IEEE Int. Conf. Robot. Autom. (ICRA), 1998.